68' MICRO JOURNAL

Australia	A $ 4.75	New Zealand	NZ $ 6.50	**$2.95 USA**
Singapore	S $ 9.45	Hong Kong	H $23.50	
Malaysia	M $ 9.45	Sweden	30:-SEK	

Motorola VME-MACINTOSH-S 50 & Other 68XXX Systems
6809 68008 68000 68010 68020 68030

OS-9 The Magazine for Motorola CPU Devices FLEX SK*DOS
A User Contributor Journal

This Issue:

Mac-Watch WETPAINT p.40
Building a GT-4 Graphic Terminal p.48
"C" User Notes p.7
Basically OS-9 p.12
PASCAL p.42
FORTH p.45 And Lots More!

IX ISSUE XI ● Devoted to the 68XXX User ● November 1987

The Grandfather of "DeskTop Publishing™"

SERVING THE 68XXX USER WORLDWIDE

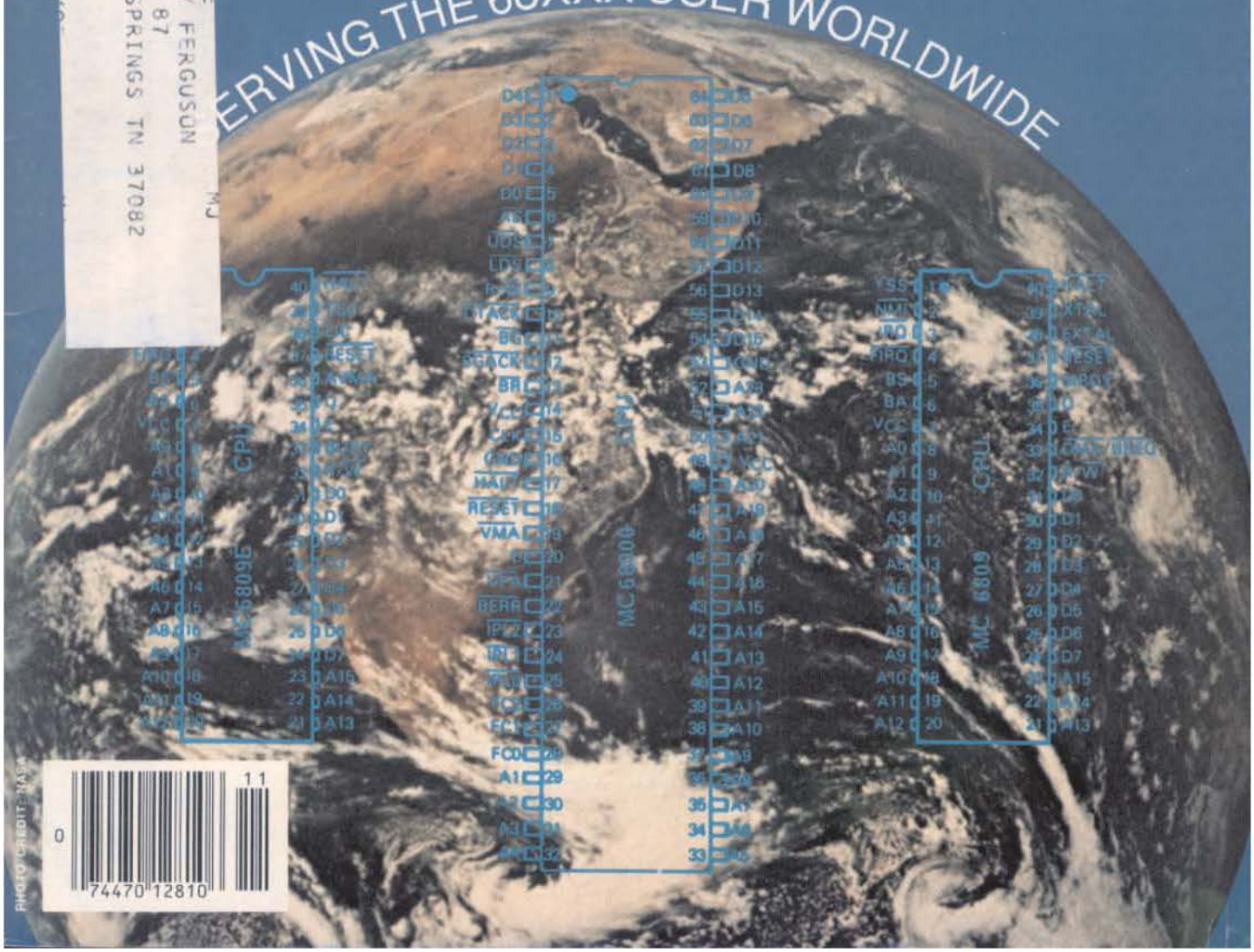
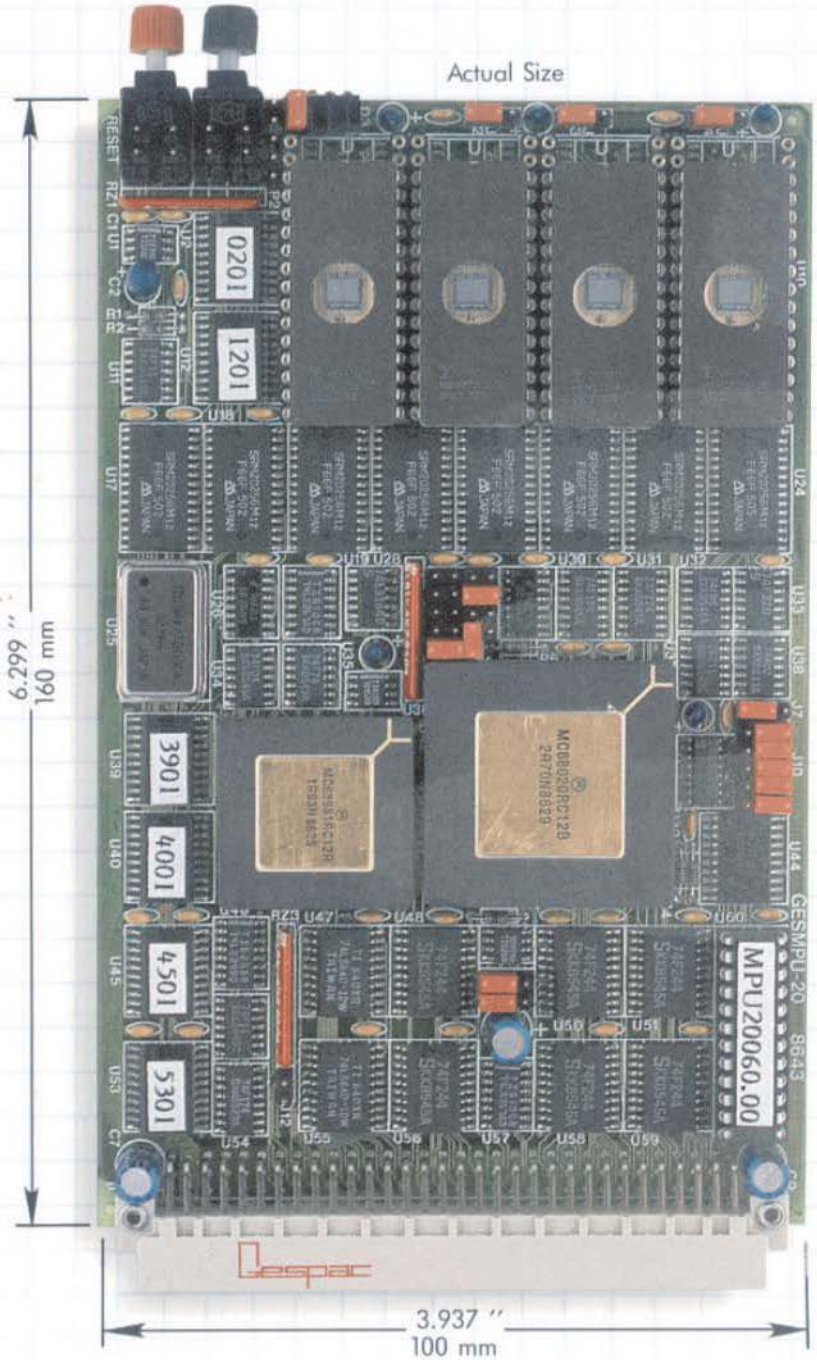


PHOTO CREDIT: NASA

0 74470 12810 11

GESPAC Gives You More Power Per Square Inch.



Here is the size, performance, and cost breakthrough you have been waiting for: The 68020 based GESMPU-20 from GESPAC.

You can now unleash an unprecedented amount of power into your applications. On just 25 square inches, we have squeezed a 12.5 MHz (16 MHz optional) 68020 32-bit microprocessor, a 68881 floating point coprocessor, 4 sockets for up to 512 Kilobytes of EPROM, and up to 512 Kilobytes of zero-wait-states CMOS RAM.

This board is totally expandable through its G-64 bus interface. And GESPAC has the largest variety of inexpensive memory, interfaces, controllers, and transducer cards anywhere. Plus real-time disk operating systems, high level languages, and other software tools. GESPAC has the total solution to your system integration needs.

Best of all, because our boards are small, they cost less. The new GESMPU-20 is priced below $1000 in one hundred piece quantity orders.

So why wait? Contact us today for information on the GESMPU-20 or any of the 150 G-64 bus system components from GESPAC—the leader in single Eurocard micr computer products worldwide.

Call (602) 962-5559.

Gespac

IN USA - CANADA
50A West Hoover Ave.
Meso, Arizona 85202
Tel. (602) 962-5559
Telex 386575

INTERNATIONAL
3, chemin des Aulx
CH-1228 Geneva
Tel. (022) 713400
Telex 429989

GMX MICRO-20 PRICE LIST

MICRO 20 (12.5 MHz) W/1 SAB $2565.00
MICRO 20 (16.67 MHz) W/1 SAB $2895.00
MICRO 20 (20 MHz) W/1 SAB $3295.00

OPTIONAL PARTS AND ACCESSORIES

68881 12.5MHz Floating Point Coprocessor ..... $ 195.00
68881 16.67MHz Floating Point Coprocessor ..... $ 295.00
68881 20MHz Floating Point Coprocessor ..... $ 495.00
MOTOROLA 68020 USERS MANUAL ..... $ 18.00
MOTOROLA 68881 USERS MANUAL ..... $ 18.00

SBC ACCESSORY PACKAGE (M20-AP) $1690.00

The package includes a PC-style cabinet with a custom backpanel, a 25 Megabyte (unformatted) hard disk and controller, a floppy disk drive, a 150 watt power supply, cooling fan, panel mounted reset and abort switches, and all necessary internal cabling. (For use with SAB—9D serial connectors only.)

2nd 5"80 FLOPPY & CABLES FOR M20-AP, ADD ..... $ 250.00
SECOND 25MB HARD DISK & CABLES, ADD ..... $ 780.00
TO SUBSTITUTE 50MB HD FOR 25MB HD, ADD ..... $ 290.00
TO SUBSTITUTE 80MB HD FOR 25MB HD, ADD $1500.00
TO SUBSTITUTE 155MB FOR 25MB HD, ADD $2100.00
60MB TEAC STREAMER WITH ONE TAPE ..... $ 870.00
PKG. OF 5 TEAC TAPES ..... $ 112.50

CUSTOM BACK PANEL PLATE (BPP-PC) ..... $ 44.00

I/O EXPANSION BOARDS

16 PORT SERIAL BOARD ONLY (SBC-16S) ..... $ 335.00

The SBC-16S extends the I/O capabilities of the GMX Micro-20 68020 Single-board Computer by adding sixteen asynchronous serial I/O ports. By using two SBC-16S boards, a total of thirty-six serial ports are possible.

RS232 ADAPTER (SAB-25, SAB-9D or SAB-8M) $165.00

The board provides level-shifting between TTL level and standard RS-232 signal levels for up to 4 serial I/O ports.

60 LINE PARALLEL I/O BOARD (SBC-60P) $398.00

The GMX SBC-60P uses three 68230 Parallel Interface/Timers (PI/Ts) to provide up to forty-eight parallel I/O lines. The I/O lines are buffered in six groups of eight lines each, with separate buffer direction control for each group. Buffer direction can be fixed by hardware jumpers, or can be software programmable for bidirectional applications.

PROTOTYPING BOARD (SBC-WW) $75.00

The SBC-WW provides a means of developing and testing custom I/O interface designs for the GMX Micro-20 68020 Single-board Computer. The board provides areas for both DIP (Dual Inline Package) and PGA (Pin Grid Array) devices, and a pre-wired memory area for up to 512K bytes of dynamic RAM.

I/O BUS ADAPTER (SBC-BA) $195.00

The SBC-BA provides an interface between the GMX Micro-20 68020 Single-board Computer and the Motorola Input/Output Channel (I/O bus). With the I/O bus, up to sixteen off-the-shelf or custom peripheral devices (I/O modules) can be connected to the GMX Micro-20.

ARCNET LAN board w/o Software (SBC-AN) $475.00

The SBC-AN provides an interface between the GMX Micro-20 68020 Single-board Computer and the ARCNET modified token-passing Local Area Network (LAN) originally developed by Datapoint Corp. The ARCNET is a baseband network with a data transmission rate of 2.5 Megabits/second. The standard transmission media is a single 93 ohm RG-62/U coaxial cable. Fiber optic versions are available as an option.

OS9 LAN Software Drivers for SBC-AN $120.00

GMX MICRO-20 SOFTWARE

020 BUG UPDATE — PROMS & MANUAL $150.00

THESE 68020 OPERATING SYSTEMS ARE PRICED WHEN PURCHASED WITH THE MICRO-20. PLEASE ADD $150.00 IF PURCHASED LATER FOR THE UPDATED PROMS AND MANUALS. ALL SHIPPED STANDARD ON 5¼" DISKS. 3½" OPTIONAL IF SPECIFIED.

OS9

OS9/68020 PROFESSIONAL PAK $850.00

Includes O.S., "C", uMACS EDITOR, ASSEMBLER, DEBUGGER, development utilities, 68881 support.

OS9/68020 PERSONAL PAK ..... $ 400.00

Personal OS-9 systems require a GMX Micro-20 development system running Professional OS-9/68020 for initial configuration.

Other Software for OS-9/68020

BASIC (Included in PERSONAL PAK) ..... $ 200.00
C COMPILER (included in PROFESSIONAL PAK) ..... $ 750.00
PASCAL COMPILER ..... $ 500.00

UNIFLEX

UniFLEX ..... $ 450.00
UniFLEX WITH REAL-TIME ENHANCEMENTS ..... $ 800.00

Other Software for UniFLEX

UniFLEX BASIC W/PRECOMPILER ..... $ 300.00
UniFLEX C COMPILER ..... $ 350.00
UniFLEX COBOL COMPILER ..... $ 750.00
UniFLEX SCREEN EDITOR ..... $ 150.00
UniFLEX TEXT PROCESSOR ..... $ 200.00
UniFLEX SORT/MERGE PACKAGE ..... $ 200.00
UniFLEX VSAM MODULE ..... $ 100.00
UniFLEX UTILITIES PACKAGE I ..... $ 200.00
UniFLEX PARTIAL SOURCE LICENSE $1000.00

GMX EXCLUSIVE VERSIONS, CUSTOMIZED FOR THE MICRO-20, OF THE BELOW LANGUAGES AND SOFTWARE ARE ALSO AVAILABLE FROM GMX.

ABSOFT FORTRAN (UniFLEX) $1500.00
SCULPTOR (specify UniFLEX or OS9) ..... $ 995.00
FORTH (OS9) ..... $ 595.00
DYNACALC (specify UniFLEX or OS9) ..... $ 300.00

GMX DOES NOT GUARANTEE PERFORMANCE OF ANY GMX SYSTEMS, BOARDS OR SOFTWARE WHEN USED WITH OTHER MANUFACTURERS PRODUCT.

ALL PRICES ARE F.O.B. CHICAGO IN U.S. FUNDS

TO ORDER BY MAIL: SEND CHECK OR MONEY ORDER OR USE YOUR VISA OR MASTER CHARGE. Please allow 3 weeks for personal checks to clear. U.S. orders add $5 handling if under $200.00. Foreign orders add $10 handling if order is under $200.00. Foreign orders over $200.00 will be shipped via Emery Air Freight COLLECT, and we will charge no handling. All orders must be prepaid in U.S. funds. Please note that foreign checks have been taking about 8 weeks for collection so we would advise wiring money, or checks drawn on a bank account in the U.S. Our bank is the Continental Illinois National Bank of Chicago, 231 S. LaSalle Street, Chicago, IL 60693, account number 73-32033.

CONTACT GMX FOR MORE INFORMATION ON THE ABOVE PRODUCTS

GMX STILL SELLS GIMIX S50 BUS SYSTEMS, BOARDS & PARTS. CONTACT GMX FOR COMPLETE PRICE LIST.

GMX 1337 W. 37th Place, Chicago, IL 60609 (312) 927-5510 — TWX 910-221-4055 — FAX (312) 927-7352

A Member of the CPI Family

68 Micro Journal

10 Years of Dedication to Motorola CPU Users

6800 6809 68000 68010 68020

The Originator of "DeskTop Publishing™"

Contents

"C" User Notes	7	Pass
Basically OS-9	12	Voigts
68XXX & the STD BUS	16	West
Logically Speaking	19	Jones
Convert an XT Keyboard to a CT82	25	Allan
Mac-Watch WETPAINT	40	Law
PASCAL	42	Reimiller
FORTH	45	Lurie
Build the GT-4 Graphic Terminal	48	Condon
Bit Bucket	53	
Classifieds	56	

68 MICRO JOURNAL

"Contribute Nothing - Expect Nothing" DMW 1986

COMPUTER PUBLISHING, INC.

"Over a Decade of Service"

"World Wide"

68 MICRO JOURNAL
Computer Publishing Center
5900 Cassandra Smith Road
PO Box 849
Hixson, TN 37343

Phone (615) 842-4600 Telex 510 600-6630

68 Micro Journal is the *original* "DeskTop Publishing" product and has continously published since 1978 using only micro-computers and special "DeskTop" software. Using first a kit built 6800 micro-computer, a modified "ball" typewriter, and "home grown" DeskTop Publishing software. None was commercially available at that time. For over 10 years we have been doing "DeskTop Publishing"! *We originated what has become traditional "DeskTop Publishing"!* Today 68 Micro Journal is acknowledged as the "Grandfather" of "DeskTop Publishing" technology.

68 Micro Journal is published 12 times a year by Computer Publishing Inc. Second Class Postage paid ISSN 0194-5025 at Hixson, TN. and additional entries. Postmaster: send form 3597 to 68 Micro Journal, POB 849, Hixson, TN 37343.

Subscription Rates

1 Year $24.50 USA, Canada & Mexico $34.00 a year. Others add $12.00 a year surface, $48.00 a year Airmail, USA funds. 2 years $42.50, 3 years $64.50 plus additional postage for each additional year.

Items or Articles for Publication

Articles submitted for publication must include authors name, address, telephon number, date and a statement that the mat rial is original and the property of the author. Articles submitted should be on diskette, OS-9, SK*DOS, FLEX, Macintosh or MS-DOS. All printed items should be dark type and satisfactory for photo-reproduction. No blue ink! No hand written articles - please! Diagrams o.k.

Please - do not format with spaces any text indents, charts. etc. (source listi go.k.). We will editi all formatti g. Text should fall flush left and use a carriage return only to indicate a paragraph end. Please write for free authors guide.

Letters & Advertising Copy

Letters to the Editor should be the original copy, signed! Letters of grip as well as praise are acceptable. *We reserve the right to reject any letter or advertising material, for any reason we deem advisable.* Advertising Rates: Commercial please contact 68 Micro Journal Advertising Department. Classified advertising must be non-commercial. Minimum of $15.50 for first 15 words. Add $.60 per word thereafter. No classifieds accepted by telephone.

MUSTANG -020 Super SBC™

DATA-COMP proudly presents the first Under $5000 "SUPER MICRO".

The MUSTANG-020™

MUSTANG-020™

The MUSTANG-020 68020 SBC provides a powerful, compact, 32 bit computer system featuring the "state of the art" Motorola 68020 "super" micro-processor. It comes standard with 2 megabyte of high-speed SIP dynamic RAM, serial and parallel ports, floppy disk controller, a SASI hard disk interface for intelligent hard disk controllers and a battery backed-up time-of-day clock. Provisions are made for the super powerful Motorola MC68881 floating point math co-processor, for heavy math and number crunching applications. An optional network interface uses one serial (four (4) standard, expandable to 20) as a 125/bit per second network channel. Supports as many as 32 nodes.

The MUSTANG-020 is ideally suited to a wide variety of applications. It provides a cost effective alternative to the other MC68020 systems now available. It is an excellent introductory tool to the world of hi-power, hi-speed new generation "super micros". In practical applications it has numerous applications, ranging from scientific to education. It is already being used by government agencies, labs, universities, business and practically every other critical applications center, worldwide, where true multi-user, multi-tasking needs exist. The MUSTANG-020 is UNIX C level V compatible. Where low cost and power is a must, the MUSTANG-020 is the answer, as many have discovered. Proving that price is not the standard for quality!

As a software development station, a general purpose scientific or small to medium business computer, or a super efficient real-time controller in process control, the MUSTANG-020 is the cost effective choice. With the optional MC68881 floating point math co-processor installed, it has the capability of systems costing many times over it's total acquisition cost.

With the DATA-COMP "total package", consisting of a heavy duty metal cabinet, switching power supply with rf/line by-passing, 5 inch DS/DD 80 track floppy, Xebec hard disk controller, 25 megabyte winchester hard disk, four serial RS-232 ports and a UNIX C level V compatible multi-tasking, multi-user operating system, the price is under $5000, w/12.5 megahertz system clock (limited time offer). Most all popular high level languages are available at very reasonable cost. The system is expandable to 20 serial ports, at a cost of less than $65 per port, in multiples of 8 port expansion options.

The system SBC fully populated, quality tested, with 4 serial ports pre-wired and board mounted is available for less that $3000. Quantity discounts are available for OEM and special applications, in quantity. All that is required to bring to complete "system" standards is a cabinet, power supply, disks and operating system. All these are available as separate items from DATA-COM .

A special version of the Motorola 020-BUG is installed on each board. 020-BUG is a ROM based bebugger package with facilities for downloading and executing user programs from a host system. It includes commands for display and modification of memory, breakpoint capabilities, a powerful assembler/disassemble and numerous system diagnostics. Various 020-BUG system routines, such as I/O handlers are available for user programs.

Normal system speed is 3-4.5 MIPS, with burst up to 10 MIPS, at 16.6 megahertz. Intelligent I/O available for some operating systems.

Hands-on "actual experience sessions", before you buy, are available from DATA-COMP. Call or write for additional information or pricing.

DATA-COMP Installed Systems World-Wide — OVER 16 YEARS OF DEDICATED QUALITY!

CPI A Division of Computer Publishing, Inc.

5900 Cassandra Smith Road
Hixson, Tn 37343
Telephone 615 842-4600
Telex 510 600-6630

Mustang-020 Mustang-08 Benchmarks

	32 bit Integer	Register Long
IBM AT 7300 Xenix Sys 3	9.7	
AT&T 7300 UNIX PC 68010	7.2	4.3
DEC VAX 11/780 UNIX Berkley_4.2	3.6	3.2
DEC VAX 11/750 "	5.1	3.2
68000 OS-9 68K 8 Mhz	10.0	9.0
68000 OS-9 68K 10 Mhz	6.5	4.0
MUSTANG-08 68000 OS-9 68K 10 Mhz	***9.8***	***6.3***
MUSTANG-020 68020 OS-9 68K 16 Mhz	***2.2***	***0.88***
MUSTANG-020 68020 MC68881 UniFLEX 16 Mhz	***1.8***	***1.22***

```
main()
    {
        register long i;
        for (i=0; i < 999999; ++i);
    }
```

Estimated MIPS - MUSTANG-020 ..... 4.3 MIPS.
Burst to 8 - 10 MIPS: Motorola Specs

OS-9

OS-9 Professional Ver. *Includes C Compiler	$850.00
Basic09	300.00
C Compiler	300.00
68000 Disassembler (w/source add: $100.00)	100.00
Pascal 77	750.00
Microware Pascal	500.00
Oregaard Pascal	900.00
Stylo-Graph	495.00
Stylo-Spell	195.00
Stylo-Merge	175.00
Stylo-Graph-Spell-Merge	695.00
PAT w/C source	229.00
JUST w/C source	79.95
PAT/JUST Combo	249.50
Sculptor+ (see below)	995.00
COM	125.00

UniFLEX

UniFLEX (68020 ver)	$450.00
Screen Editor	150.00
Sort-Merge	200.00
BASIC/PreCompiler	500.00
C Compiler	350.00
COBOL	750.00
CMODEM w/source	100.00
TMODEM w/source	100.00
X-TALK (see Ad)	99.95
Cross Assembler	50.00
Pascal 77	450.00
Sculptor+ (see below)	995.00

Standard MUSTANG-020™ shipped 12.5 Mhz.

Add for 16.6 Mhz 68020	375.00
Add for 16.6 Mhz 68881	375.00
Add for 20 Mhz 68020/RAM	750.00
16 Port exp. RS-232 requires 1 or 2 Adapter Cards below.	335.00
RS232 Adapter Each card supports 4 additional ser. ports (total of 36 serial ports supported)	165.00
60 line Parallel I/O card Uses 3 68230 Interface/Timer chips, 6 groups of 8 lines each, separate buffer direction control for each group	398.00
Prototype Board areas for both dip and PGA devices & a pre-wired memory area up to 512K DRAM.	75.00
SBC-AN Interface between the system and ARCNET modified token-passing LAN, fiber optics optional - call. LAN software drivers	475.00 120.00
Expansion for Motorola I/O Channel Modules	$195.00

Special for complete MUSTANG-020™ system buyers - Sculptor+ $695.00. SAVE $300.00

Software Discounts

All MUSTANG-020™ system and board buyers are entitled to discounts on all listed software: 10-70% depending on item. Call or write for quotes. Discounts apply after the sale as well.

THE PRO!

Only the "PRO" version of OS-9 supported!

This is *HEAVY DUTY* Country!

UPGRADES
Write or Call for Professional OS-9 "Full Bore" Upgrade Kit

For a limited time we will offer a $400 trade-in on your old 68XXX SBC. Must be working properly and complete with all software, cables and documentation. Call for more information.

12.5 Mhz (optional 16.6 Mhz available) MC68020 full 32-bit wide path
32-bit wide data and address buses, non-multiplexed
on chip instruction cache
object code compatible with all 68XXX family processors
enhanced instruction set - math co-processor interface
68881 math hi-speed floating point co-processor (optional)
direct extension of full 68020 instruction set
full support IEEE P754, draft 10.0
transcendential and other scientific math functions
2 Megabyte of SIP RAM (512 x 32 bit organization)
up to 256K bytes of EPROM (64 x 32 bits)
4 Asynchronous serial I/O ports standard
optional to 20 serial ports
standard RS-232 interface
optional network interface
buffered 8 bit parallel port (1/2 MC68230)
Centronics type pinout
expansion connector for I/O devices
16 bit data path
256 byte address space
2 interrupt inputs
clock and control signals
Motorola I/O Channel Modules
time of day clock/calendar w/battery backup
controller for 2, 5 1/4" floppy disk drives
single or double side, single or double density
35 to 80 track selectable (48-96 TPI)
SASI interface
programmable periodic interrupt generator
interrupt rate from micro-seconds to seconds
highly accurate time base (5 PPM)
5 bit sense switch, readable by the CPU
Hardware single-step capability

Mustang Hi-Speed Systems
Only from Data-Comp Div.
68008-68030

Don't be mislead! ONLY Data-Comp delivers the Super MUSTANG-020

These hi-speed 68020 systems are presently working at NASA, Atomic Energy Commission, Government Agencies as well as Universities, Business, Labs, and other Critical Applications Centers, worldwide, where speed, math crunching and multi-user, multi-tasking UNIX C level V compatibility and low cost is a must!

MUSTANG-020 SBC		$2498.00
Cabinet w/switching PS		$299.95
5"-80 track floppy	DS/DD	$269.95
Floppy cable		$39.95
OS-9 68K Professional Ver. * Includes C Compiler ($500.00)		$850.00
Winchester cable		$39.95
Winchester Drive 25 Mbyte		$895.00
Hard Disk controller		$395.00
Shipping USA UPS		$20.00
Total:	Save $1000.00	$5,299.80
complete sytem		$4,299.80

UniFLEX	Less	$100.00
MC68881 I/p math processor	Add	$275.00
16.67 Mhz MC68020		$375.00
16.67 Mhz MC68881		$375.00
20 Mhz MC68020 Sys		$750.00

Note all 68881 chips work with 20 Mhz Sys

NOTE: Only Professional OS-9 now available (68020 Version) Includes ($500.00) C Compiler 68020 & 68881 supported

complete
25 Mbyte HD System
$4299.80
85 Mbyte HD System
$5748.80

Data-Comp Division
A Decade of Quality Service
Systems World-Wide
Computer Publishing, Inc. 5900 Cassandra Smith Road
Telephone 615 842-4601 - Telex 510 600-6630 Hixson, Tn 37343

PAT - JUST

PAT

With 'C' Source

$229.00

All OS-9 68XXX Systems

PAT FROM S. E. MEDIA -- A FULL FEATURED SCREEN ORIENTED TEXT EDITOR with all the best of PIE. For those who swore by and loved **PIE**, this is for **YOU! All PIE features & much more!** Too many features to list. And if you don't like ours, change or add your own. **C source included.** Easily configured to your CRT terminal, with special configuration section. No sweat!

68008 - 68000 - 68010 - 68020 OS-9 68K $229.00

COMBO PAT/JUST

Special $249.00

JUST

JUST from S. E. MEDIA - - Text formatter written by Ron Anderson; for dot matrix printers, provides many unique features. Output formatted to the display. User configurable for adapting to other printers. Comes set-up for Epson MX80 with Graflex. Up to 10 imbedded printer control commands. Compensates for double width printing. Includes normal line width, page numbering, margin, indent, paragraph, space, vertical skip lines, page length, centering, fill, justification, etc. Use with **PAT** or any other text editor. The ONLY stand alone text processor for the 68XXX OS-9 68K, that we have seen. And at a very **LOW PRICE!** Order from: S.E. MEDIA - see catalog this issue.

68008 - 68000 - 68010 - 68020 OS-9 68K
With 'C' source $79.95

The C Programmers Reference Source. Always Right On Target!

C User Notes

A Tutorial Series

By: Dr. E. M. 'Buf' Pass
1454 Latin Lane N.W.
Conyers, GA 30207
404 483-1717/4570
Computer Systems Consultants

INTRODUCTION

This chapter covers some of the C reference materials currently available for those attempting to learn how to use the C language or to learn how to become more proficient in its application.

Bill Gates, President of Microsoft, recently was quoted in the trade press as saying that most major technical and development software for micro and mini computers is either now written in the C language or is being rewritten into the C language.

Also in the trade press was the announcement by Lotus that the best-seller 1-2-3 package had been rewritten into C, would be available very soon under UNIX and VMS and would be available on IBM-compatible mainframes next year.

C REFERENCES

Most C reference materials use the PC, UNIX, or VMS systems in their expositions. For those interested in learning C for OS/9, UNIFLEX, or FLEX, these references may be used with a greater or lesser degree of success. Depending upon the nature of the subject, the more specific and advanced references may well be essentially effectively useless for these operating systems. For example, graphic and window software for the PC may be of little use on hardware radically different from that on the PC. However, it is necessary and useful to describe the materials, because many are interested in C for several different systems.

Since any list of reference materials is necessarily incomplete, many readers will note favorite C texts not included. If readers will send very short reviews of such texts, I will consider including them in the future.

The C Programming Language
B. Kernighan and D. Ritchie

Although this book is somewhat dated, obscure, and vague, it is still recognized as the standard text for the C language. Every C programmer should have a copy and should have actually read it at least once, to better gain a feeling for the flavor of the C language, from the original source. However, most programmers will need much more than this book to help to help them use the C language proficiently.

The C Answer Book
Tondo and Gimpel

This book is a follow-up to the original white book (K and R). It reprints each exercise from the K and R book, along with a complete answer and explanation for it. The index assists the C programmer in locating particular topics in the exercises.

Programming in C
K. A. Jamsa

This book presents a building-block approach intended to help the reader in learning the C language. Beginning with simple concepts, it introduces more complex concepts in each chapter. It covers data types, program structure, string processing, pointers, arrays, and buffered and unbuffered input/output. It also discusses the use of several UNIX utilities and tools.

The C Library
K. A. Jamsa

This book provides over 125 structured and commented C library and utility routines. These vary from simple macros and functions to complete input/output subroutines, primarily for UNIX or MSDOS. Each library

routine provides comments describing its function, argument list, routines called, and control flow. These routines illustrate the following parts of the language: file manipulation, sorting, recursion, input/output, string manipulation, pointers, and arrays.

The C Programmer's Handbook
AT&T Bell Laboratories
M. I. Bolsky

This book is formatted as a handbook providing a topic-oriented introduction and reference for the C language. Since the author is an employee at the AT&T Systems Training Center, the handbook covers only the UNIX System 5 C language and its usage on UNIX System 5 systems. In addition to covering the C language itself, this book provides information on portability, covering compiler and machine dependencies, program organization, data file conversion, and other portability issues.

C: An Advanced Introduction
AT&T Bell Laboratories
N. Gehani

This book provides an introduction to the C language for those already familiar with a procedural language such as FORTRAN or ALGOL. It emphasizes the advanced features of the language, such as type declarations, data abstraction, exception processing, concurrent programming, the C preprocessor, and programming tools. The author assumes that the reader will be using a UNIX-based system for working the problems and for later development.

C: The Complete Reference
H. Schildt

This book is organized as an encyclopedia of C terms, functions, codes, applications, etc. It is divided into the following five parts: review material, C libraries, algorithms and applications, efficiency and portability, and C++. It also provides information about the new ANSI X3-J11 Draft Proposed C Standard.

C Made Easy
H. Schildt

This book is designed for Basic programmers wanting to learn the C language. It provides a tutorial on the C language, in addition to side-by-side examples of Basic programs and their C translations. It also provides a description of common C programming errors and how to avoid making them.

Solutions in C
R. Jaeschke

This book contains several hundred tips and guidelines for programming in the C language. It is reference material for experienced C programmers and provides suggestions for how best to engineer a programming project to make best use of the C language. It covers the following subjects: structures, bit fields, unions, arrays, data storage, headers, C preprocessor, definition of, reference to, and pointers to functions, the standard C run-time library, etc. It also provides information about the new ANSI X3-J11 Draft Proposed C Standard.

Programming in C
S. Kochan

This book illustrates the usage of the C language with about 100 C programs and functions. Each program or function is structured and commented to demonstrate particular points about the language. The book covers the following topics: looping, decisions, arrays, functions, structures, strings, bit operations, and modularity. Appendices to the book summarize the syntax of the language and list common programming mistakes to avoid.

C Programming Guide (Second Edition)
J. Purdum

This book provides a tutorial for beginners desiring to learn to use the C language. It offers tips, tricks, and techniques used by C experts. It explains how the language works and compares several C programs to equivalent Basic programs. It also provides information about the new ANSI X3-J11 Draft Proposed C Standard.

C Programmer's Library
J. Purdum, T. Leslie, A. Stegemoller

This book presents C programming techniques exemplified with the following C programs and routines: disk sort, terminal library, installation program, ISAM library, and book cataloger. These C programs and routines are usable as presented or may be modified for special purposes. The book provides machine and compiler specific information as a comment with each listing. The programs and routines are also available on diskette for MSDOS and CP/M.

C Self-Study Guide
J. Purdum

This book covers many aspects of C programming with a question and answer format. It addresses the usage of C at many levels, from the beginner's problems with syntax errors to the expert's usage of the more advanced features and nuances of the language and of the standard C library routines.

C: A Reference Manual (Second Edition)
S. P. Harbison and G. L. Steele

In addition to presenting a complete outline and summary of the C language, this book discusses the evolving state of the C language, including a summary of the new ANSI X3-J11 Draft Proposed C Standard. The authors present the C language in a bottom-up order: lexical structure, preprocessor, declaration types, expressions, statements, functions, programs, and standard run-time libraries. The book also covers over 180 of the standard run-time library routines in detail.

Debugging C
R. Ward

This book discusses the difficulties involved in debugging C programs and suggests better methods for accomplishing this task. It is not a book on avoiding common C programming mistakes, although most experienced C programmers should be able to derive new techniques for writing C programs which assist in the debugging process. It provides a carefully-defined and scientific-method-based series of techniques, in an attempt to structure this debugging process. Although the book is oriented towards debugging programs written on and for MSDOS and UNIX, most of the techniques presented are relevant for debugging programs on other computers and operating systems.

Advanced C
H. Schildt

This book attempts to assist experienced C programmers in developing advanced skills in the use of the C language. It is essentially a discussion of the implementation of data structures in C. It covers the following topics in this area: operating system interface, compressed data formats, memory allocation, linked lists, binary trees, sorting and search techniques, stacks, queues, and data encryption. It also covers program and data portability, simulations, debugging, and converting Basic and Pascal programs to C.

System V Interface Definition Vol. 1-3
AT&T Information Systems

This series of books provides a rather complete definition of the UNIX System 5 operating system, system calls, library, C compiler, and utilities. Although it is probably more detailed than most C programmers require, it is a source of information for those needing to port programs to or from UNIX System 5 and clones, or to port UNIX System 5 to a new environment.

Operating Systems: Design & Implementation
A. S. Tanenbaum

This book provides a discussion of the design and implementation of operating systems, primarily in terms of an example operating system called MINIX. The specifications for MINIX derive from UNIX Version 7. Not only is the source code (in C, of course) printed in the book, but source and object versions of MINIX are available for the PC on diskette. In addition to the commented source code for MINIX and explanations of how it works, the book provides a lengthy discussion of four areas of operating system theory: process management, input/output management, memory management, and file system management, and how they relate to MINIX and to other operating systems.

Portable C and UNIX System Programming
J. E. Lapin

This book summarizes the similarities and differences among the various versions of UNIX C languages. It covers all commands, library functions, and system calls available on UNIX System 5, Berkeley UNIX 4.2 and 4.3, Microsoft Xenix, and SCO Xenix 5.0, in addition to those specified in the System V Interface Definition and in the proposed ANSI X3-J11 Standard C Language.

Applied C
Strawberry Software, Inc.
edited by B. Derman

This book is composed of over twenty chapters, each written by an expert in a particular area. It covers the generation of program specifications, the consideration of user interfaces, the aspects of programming style, portability, modularity, data structures, language parsing, indexing, and screen management, methods of using compilers, linkers, loaders, and libraries, and usage of utilities such as make, lint, etc.

The C Puzzle Book
A. R. Feuer

This book provides a series of topics, each of which is composed of C programs or functions representing a puzzle about some aspect of the C language and its usage. These puzzles, with solutions, often pertain to little-used features of the language, and are intended to force the reader to think about parts of the language which they may never have thought about before.

The C Trainer
A. R. Feuer

This book and diskette comprise a tutorial for C in a structured programming environment. The diskette contains a C interpreter and series of C program and function examples to accompany the book. The following advanced topics are covered (in addition to more basic topics): co-routines, binary trees, dispatch tables, indirect pointers, syntax parsing, finite-state automata, semantic networks, and recursion.

Programming Pearls
J. Bentley

This book consists of a series of essays pertaining to practical aspects of programming and computer science, originally published in the Communications of the Association for Computing Machinery. It discusses common-sense engineering techniques and prototyping as they relate to computer programming. Many of the problems may best be solved or improved by using insight into the nature of the situation, rather than the obvious solution.

C Programmer's Reference Card
Two Colors

This quick-reference card summarizes all of the commonly-used aspects of C, provides usage of programming-related UNIX utilities, and contains and an ASCII code chart, all on a plastic sheet.

C Language Poster
AGS

This wall poster provides a guide to C language syntax and usage.

C PROBLEM

The following fragment of a C program works correctly on some systems but fails on other systems:

```
#include <stdio.h>
:
main(argc, argv)
int argc;
char *argv[];
{
    :
    if ((argc > 5) && (*argv[5] == 'x'))
        strcpy(argv[5], "last argument");
    :
}
```

The problem lies in the copying of characters into the memory pointed to by argv[5]. If the space allocated for the string is shorter than 14 characters, the memory just beyond the space allocated will be overwritten. In this case, the memory space overwritten may be exceptionally critical, since it will probably involve stack space beyond the point at which the program was executed, because the second argument to the "main" function points to the command-line arguments.

Many solutions are possible; following is one possible method. Most, like this one, involve changing the pointer to the argument.

```
#include <stdio.h>
:
main(argc, argv)
int argc;
char *argv[];
{
    :
    if ((argc > 5) && (*argv[5] == 'x'))
        argv[5] = "last argument";
    :
}
```

Another class of solutions involves copying the pointers to the arguments into another array. Following is one solution in this class:

```
#include <stdio.h>
:
main(argc, argv)
int argc;
char *argv[];
{
    char arg[10][32], *strncpy();
    int a;
    :
    if (argc > 10)
        argc = 10;
    for (a = 0; (a < argc); ++a)
        *(strncpy(arg[a], argv[a], 31) + 31) = 0;
    if ((argc > 5) && (*arg[5] == 'x'))
        strcpy(arg[5], "last argument");
    :
}
```

EXAMPLE C PROGRAM

Following is this month's example C program; it scans the contents of a list of files, extracting printable strings from each no shorter than a specified length. It could be used on a text file, with its output piped to a program to sort and remove duplicate strings, to study the words used in the text. It could also be used on a executable file, in an attempt to locate help or prompt strings, which might assist in using, understanding, or disassembling the program.

```
#include <stdio.h>
#include <ctype.h>

#define STRING 80
#define BUFFER 512
```

```
#ifdef IBMPC
#define BINARY "rb"
#endif
#ifdef FLEX
#define BINARY "rb"
#endif
#ifndef BINARY
#define BINARY "r"
#endif

main(argc, argv)
int argc;
char **argv;
{
    FILE *input = stdin;
    int enough = 3;
    char *prog = argv[0];

    while (argc > 1)
    {
        if (*argv[1] == '-')
        {
            enough = 0;
            sscanf(argv[1] + 1, "%d", &enough);
            if ((enough < 1) || (enough > (STRING -
3)))
            {
                        printf("\n\nString Search
Program\n\n");
                    printf("Usage: %s [-n] [file-
list]\n\n",
                   prog);
                 printf("finds strings of length >=
n\n");
               printf("(default 3) in file-list\n");
               printf("(default stdin)\n\n");
               exit(1);
            }
        }
        else
        if (!(input = fopen(argv[1], BINARY)))
        {
             printf("%s: Cannot find %s\n", prog,
argv[1]);
            exit(1);
        }
        else
            strings(input, enough);
        --argc;
        ++argv;
    }
    exit(0);
}

strings(input, enough)
FILE *input;
int enough;
{
    int chars, i, n;
    char buffer[BUFFER], c, *p, string[STRING];

    while (n = fread(buffer, 1, BUFFER, input))
    {
        for (p = buffer; n; --n)
        {
            c = *p++;
            if (isprint(c))
            {
                string[chars++] = c;
                if (chars > (STRING - 3))
                {
                    string[chars++] = '\\';
                    string[chars] = 0x00;
                    printf("%s\n", string);
                    chars = 0;
                }
            }
            else
            {
                if ((chars >= enough) && ((!c) ||
                    (c == '\n')))
                {
                    string[chars] = 0x00;
                    printf("%s\n", string);
                }
                chars = 0;
            }
        }
    }
    if (input)
        close(input);
}
```

EOF

Basically OS-9

Dedicated to the serious OS-9 user.
The fastest growing users group world-wide!
6809 - 68020

A Tutorial Series

By: Ron Voigts
2024 Baldwin Court
Glendale Heights, IL

KEEPING TRACK OF OS-9 MODULES

Last month, we went into why programs refused to get into memory. There were a number of items that could cause problems. The module's attributes, CRC and header parity had to be correct. If these all were in order, there was a good chance that when an attempt was made to run the program it would get loaded into memory. Then it could be run. *OS-9 does not blindly plunk the module in memory. Rather it places it and keeps track of vital information.*

A module directory maintained. Not everything that gets into the module directory has to be executable. Device descriptor and data modules cannot be executed, but none the less they are put into the directory. But everything that gets into the directory must meet the previous criteria.

The module directory keeps track of two important pieces of information. Where the module can be found and what its link count is. Surprising at first is the fact that its name does not get entered. A little thought on it will show that once a module's location is known, other things of interest can be found. These include its name, size, language type and so forth. All that is necessary is to search the directory, for the module's location.

Level I OS-9 keeps module directory entries in 4 byte chunks. My system provides 256 bytes for the directory. This comes to 64 possible entries. I have never filled it up. The directory's location can be found from the 4 bytes stored at D.ModDir in the Direct Page Variables. These are values located in the first 256 bytes of memory. D.ModDir is located at absolute address $0026. The four bytes in my system located here are:

$03 00 04 00

This means that the directory module starts at memory location $0300. It is one page or 256 bytes long. The $0400 shows were it ends. In reality the end is at $03FF. In programming, using the $0400 is a simple way to detect when the directory's end has been past.

Entries are in 4 byte chunks. The first two bytes are the module's location. The next is its link count. And the last one is a dummy. It doesn't do anything. But 4 is easier to program with when using a loop.

It is easiest to demonstrate the table with an example. Listing 1 is a BASIC09 program that will generate a table that represents the module directory. The program gets the start and finish values for the directory. It moves through the directory extracting the modules location and link count. The dummy byte is ignored. Also where a location comes up that is $0000, it is not printed. This is not the location of a module, but rather the a null entry into the table. I ran this program and here is the result.

```
Addr  Link
----  ----
F05D    00
F852    01
FD46    01
FD74    01
B600    01
B633    00
```

This is only the first few entries. It shows the modules's locations and their link count. These entries are part of my OS9 system.

To see what these really mean, we must look at each location. We can find the module name, its size, type, revision, and name. Good news! There is a command to do this for us. It is MDIR. If I enter:

OS9: mdir e

This tells MDIR to do an extensive module directory search. The result is:

```
ADDR SIZE TY RV AT UC   NAME
---- ---- -- -- -- -- ----------
F05D  7E7 C1  1 r     OS9
F852  4F4 C1  1 r   1 OS9p2
FD46   2E C0  1 r   1 Init
FD74  17E C1  1 r   1 Boot
B600   33 F1  2 r   2 D0
B633   33 F1  2 r   2 D1
```

This goes on, but is show what the modules really are. MDIR reads the module directory and then goes to the memory location and reads the balance of the information from the module header. It can be seen that the addresses of modules in the directory are used to find them. So, $F05D with a link count of 00 is OS9, $f852 with link count of 1 is qOS9P2 and so forth.

Level 2 has a module directory that is a bit more complicated and involved. It uses 8 bytes per entry. They are 2 byte entries for MD$MPDAT, MD$MBSiz, MD$MPtr, and MD$Link.

MD$MPDAT is the Module DAT Image pointer. This is a DAT pointer to the starting of the block where the module is stored. The block is 4K long on SS-50 buses and 2K on system with a 6829 MMU. This memory location is a special one that is not necessarily associated with a specific process. Later, when the module is executed, it will be mapped into the particular task's area.

Next is MD$MBSiz. This is the actual amount of amount of the block used. Even if a small module is loaded, it will get the entire block. This is why it pays to merge modules together. Then they will be located in a continuous area of memory.

MD$PTr is a pointer to the module within the block. It is the offset from the start of the block to where the module is located. This probably would be $0000. But where numerous modules have been entered, they each will have there own offset within the block.

Finally there is the module link count. MD$Link is the link count of the module. As long as it is 1 or more, it will occupy the memory block that has been allocated to it. When it goes to zero, OS-9 will remove it and put the memory back into the memory pool. A nice result of this is if a number of modules have been loaded into the same memory block, they all will remain, as long as one is linked.

Listing 2 is Basic09 program that will print the module directory for you. It differs from the Level 1 version. Not only is the directory format different as I have outlined, but the memory cannot be PEEKed as was done before. Have no fear, there is a way.

The Level 2 System Call, F$GModDr, can be used. It copies the module directory into a 2k buffer, supplied by the caller. It also returns the start and end of the module directory in the system memory. On entry register X points to the buffer. On return register U is the start of the memory address and Y is the end. Now, how do we use it from BASIC09?

Easy! The BASIC09 TOOLS available from Southeast Media have a module called SYSCALL. This module written in 6809 object code provides a means to pass parameters to the CPU's registers, execute one of the OS-9 Service Requests and return the information in the registers. This makes it an easy task to access the service request from BASIC09.

The call to SYSCALL requires two variables to be passed. One is a complex variable, 10 bytes in size. It contains the all the 6809 CPU registers, except for S, the stack pointer. The other is the numerical value of the system call. This routine requires checking the returned values for successful execution. Notice how the program in Listing 2 checks whether the LSB of REG.CC is set. If it is set, then REG.B has the error code number.

Enough said about the program. Let's run it and see what it does.

```
ENTRY# IMAGE  SIZE   MODULE LINK
------ ------ ------ ------ ------
0001   0FE8   1000   0000   0001
0002   0FF8   0D1B   0A40   0001
0003   0FE8   1000   02BC   0001
0004   0FE8   1000   051D   0001
0005   0FE8   1000   0970   0000
0006   0F22   3FC5   0400   0001
0007   0F22   3FC5   10A9   0001
```

Again I am only showing a partial listing of the programs output. But it does display the information from the module directory. We can take a look at one of the memory blocks. Notice that DAT Image Pointer $0FE8 has 4 modules in it. The entire block in 4096 ($1000) bytes in size. The modules are identified by their offsets from the start of the block. Entry $0001 is the first one at offset $0000. Entry $0002 is at $02BC, $0004 is at $051D and $0005 is at $0970. It is a fairly good bet that this last one ends at $0FFF, recalling the entire block size is $1000 bytes.

There is an easier way to see the contents of module directory. Again, it is MDIR. The Level 2 OS-9 uses a different version of MDIR. This one uses the service request, F$GModDr. It prints out similar information, like the Level 1 version. But it is tuned to the Level 2 system requirements. Using is on my system, gives us:

```
Block Offset Size Typ Rev Attr  Use Module Name
----- ------ ---- --- --- ----  --- -----------
   FE      0  2BC  C1   F r...    1 Boot
   FD    A40  2DB  C1     r...    1 Boot3
   FE    2BC  261  C1     r...    1 Boot1
   FE    51D  2D3  C1     r...    1 Boot2
   FE    970  690  C0   8 r...    0 OS9p1
    1    400  CA9  C0   2 r...    1 OS9p2
    1   10A9   2E  C0   1 r...    1 Init
```

Here we see block $FE corresponds to the DAT Image Pointer at $0FE8.The 4 modules it contains are located at the offsets that were described earlier. If we take the last module and add the size to it, we find that it ends at $0FFF. This makes the block of memory used 4096 bytes long as pointed out earlier.

These two listings give you a means to look at the module directory. It is more practical to use the MDIR. But it is fun to dissect the system and see what makes it work.

Before I totally leave the subject of this months programs, I want to say one more thing about the BASIC09 TOOLS. If you do have the package, a few lines in Listing 1 can be replaced. Specifically lines $01BD, $01D6 and $0240. The TOOLS have a subprogram called DPEEK. It returns the integer value at a memory location. Using it, line $0240 would be changed to:

RUN DPEEK(MD.location,index)

The other lines would be changed accordingly. Either method works. There are a total of 21 subprograms in the BASIC09 TOOLS Besides the fact that I created them, they make a nice addition to your BASIC09 library.

LINK IT TOGETHER

I could not leave this month without mentioning LINKing to modules. It differs between Level 1 and 2 somewhat. And it is worthwhile mentioning.

In Level 1, linking to a module is rather simple. Whenever a module is linked to a process, the link count is increased. This insures that is will remain in memory, as long as someone is using it. A general rule to follow is only UNLINK what you have LINKed. If you cause the link count to go to zero, then it will be removed from memory. This could be very upsetting for another process. This is all that Linking does for Level 1.

Level 2 systems also use the link count as an indicator of how many processes are using a particular module. But it does a little more. Linking to a module maps it into the task's space. The module does not move in physical memory. Rather it appears to be located in the processes memory space. This is the illusion of Dynamic Address Translation. When the module is unlinked, its count is decreased and it disappears from the memory of the process.

Generally, linking is accomplished with a call to the F$LINK. This system service request requires two input parameters. Register X points to the module name. Register A contains the module type and language byte. On returns the register Y points to the entry address for the module. U points to the start of the module. And, of course, the link count is increased.

Most times you will not have concern yourself with using F$LINK. When a module is loaded or executed, it is linked. these all occur automatically and should not be of major concern.

That raps up this month. I hope I have given you some food for thought. Until next time, have a good month.

```
LISTING 1

PROCEDURE GetModDir
0000        (* ********************
0018        (*
001B        (* Name: GetModDir
002D        (* By: Ron Voigts
003E        (* Date: 2-AUG-87
004F        (* Version: 1.00
005F        (*
0062        (* ********************
007A        (*
007D        (* Function:
0089         (* This Basic09 program returns the
module
00B3         (* directory for leve 1 OS-9.  It
```

```
prints
    00DB        (* the information in tabular form.
    00FE        (*
    0101        (* ********************
    0119
    011A        (* Module directory pointers
    0136                                        TYPE
module_directory=start,finish:INTEGER
    0145        DIM ModDir:module_directory
    014E        DIM D_ModDir:INTEGER
    0155
    0156        (* Module Directory Enty Variables
    0178          TYPE md_entry=location:INTEGER;
link,dummy:BYTE
    018D        DIM MD:md_entry
    0196        DIM index:INTEGER
    0190
    019E
    019F        (* Intialize variables
    01B5        D_ModDir:=$26
    01BD
ModDir.start:=PEEK(D_ModDir)*256+PEEK(D_ModDir+1)
    01D6
ModDir.finish:=PEEK(D_ModDir+2)*256+PEEK(D_ModDir+3
)
    01F2
    01F3        (* Print a header
    0204        PRINT "Addr  Link"
    0212        PRINT "----  ----"
    0220
    0221        (* Print the table
    0233        index:=ModDir.start
    023E        WHILE index<ModDir.finish DO
    024E
MD.location:=PEEK(index)*256+PEEK(index+1)
    0267          MD.link:=PEEK(index+2)
    0277          IF MD.location<>0 THEN
    0286                          PRINT  USING  "h4,'
',h2",MD.location,MD.link
    02A4          ENDIF
    02A6          index:=index+4
    02B1        ENDWHILE
    02B5        END
```

LISTING 2

```
PROCEDURE GetModDir
    0000        (* ************************
    001C        (*
    001F        (* Name: GetModDir
    0031        (* By: Ron Voigts
    0042        (* Date: 24-JUL-87
    0054        (* Version: 2.00
    0064        (*
    0067        (* ************************
    0084        (*
    0087        (* Function:
    0093         (* This Basic09 program returns the
module
    00BD        (* directory for level 2 OS-9.
    000B          (* It prints the information in
tabular
    0102        (* form.
    010A        (*
    010D        (* ************************
    012A        (*
    012D        (* Outside procedures required:
    014C        (* SYSCALL from BASIC09 TOOLS
    0169        (*
    016C        (* ************************
    018A        (*
    0180
    018E        (* Module Directory Entry Variables
    01B1                                        TYPE
md_entry=MPDat,MBSiz,MPtr,Link:INTEGER
    01C8        DIM MD(256):md_entry
    01D6
    01D7        (* 6809 Register Variables
    01F1           TYPE  registers=cc,a,b,dp:BYTE;
x,y,u:INTEGER
    0216        DIM reg:registers
    021F
    0220        (* Level II Functions Service Request
    0245        DIM GModDr:INTEGER
    024C        GModDr:=$1A
    0254
    0255        (* Indexes for later use
    026D        DIM index,i:INTEGER
    0278
    0279        (* Output path
    0287        DIM path:INTEGER
    028E
    028F
    0290        (* Call to F$GModDr
    02A3        reg.x:=ADDR(MD)
    02B1        RUN syscall(reg,GModDr)
    02C0
    02C1          (* Trap any errors that may have
occurred
    02EA        IF MOD(reg.cc,2)=1 THEN
    02FC          ERROR reg.b
    0304        ELSE
    0308
    0309             (* Now we write the output to
standard output
    0336          PRINT "ENTRY# IMAGE  SIZE   MODULE
LINK"
    035A          PRINT "------ ------ ------ ------
------"
    0380          index:=1
    0387          FOR i:=reg.x TO reg.y-8 STEP 8
    03A7                                  PRINT  USING
"S(X1,H4,X2)",index,MD(index).MPDat,MD(index
).MBSiz,MD(index).MPtr,MD(index).Link
    03E3            index:=index+1
    03EE          NEXT i
    03F9        ENDIF
    03FB        END
```

EOF

68XX(X) And The STD BUS

Bill West
BILL WEST INCORPORATED
174 Robert Drive
Milford, Connecticut 06460
203 878-9376

This is the first of what will hopefully be a series of articles describing various hardware configurations of 68XX(X) systems well-suited to industrial and/or control type applications. This includes systems that must operate in harsh environments, ROM-based systems, and any system that must interface to something other than a standard CRT, printer, or modem. Since I primarily work with OS9(tm) systems, I will tend to describe the systems from that perspective. The intent of the articles is to familiarize readers of 68 Micro with various types of systems, and not to provide comprehensive or complete technical specifications for any of the systems presented. Any reader interested in using any of the systems will need to obtain detailed specifications from the supplier of the system, or you can contact me for more detailed information. The first several articles will concentrate on the STD bus. Later articles will describe how to use Macintosh(tm) and Atari ST(tm) computers in control applications.

The STD bus is a simple, inexpensive bus structure that is well-suited to the requirements of many control-type systems. A wide variety of boards are available, including those you would expect such as CPUs and memory boards, and some strange beasts such as those with pneumatic valves or radio receivers mounted on board. The STD bus was originally developed by Pro-Log Corporation as a simple bus structure for eight bit systems, and was released for unrestricted use with no trademarks, copyrights, or patents. STD bus circuit cards are 6.5" by 4.5" with a 56 contact card edge connector along one of the short sides. The opposite short side is considered the user interface, with whatever connections are required for the functions of a specific card. The two long edges are supported by card guides. This provides a stable mounting configuration with the card supported on two sides by card guides and on a third side by the backplane connector, allowing STD bus systems to be used for applications subject to shock and vibration.

The STD bus as originally conceived was designed to support typical eight bit processors, and the Z80 in particular. The 56 fingers of the backplane connector include 16 address lines, 8 data lines, 22 control lines, 6 logic power lines, and 4 auxiliary power lines. The processor has access to a 64 Kbyte memory address space and a 256 byte page of I/O address space. In addition, memory expansion and I/O expansion control lines allow access to an additional 64 Kbytes of memory and a second 256 byte page of I/O. Recent variations of the STD bus have multiplexed data and address lines to handle 20 bit addressing, with some proposals for 24 bit addressing and 16 bit data transfers. A number of 8088 CPU boards using 20 bit addressing and eight bit data transfers are available. These newer CPUs allow sixteen bit addressing of I/O devices. Most of the 68008 boards available on the STD bus attempt to conform to the 8088 specification to a greater or lesser extent. The definition of the control lines on the STD bus is processor dependent, with various peripheral cards designated as STD-Z80, STD-65/68xx, STD-8088, or STD-NSC800. This causes some compatibility problems between cards of different flavors, but a little reprogramming with a soldering iron and X-acto knife will handle most problems.

The STD bus is a simple, inexpensive bus structure that is well-suited to the requirements of many control-type systems...

The biggest attraction of the STD BUS is the wide variety of I/O cards which are available...

Let me make a slight digression here for those die-hard Motorola users who have never heard of Intel or the Z80. Unlike Motorola processors, which treat I/O devices just like memory locations, Intel processors and the Z80 have special instructions for accessing I/O ports. A control line from the processor indicates whether a particular bus access is a memory access or an I/O port access, and the bus cycle timing is usually different for I/O accesses. The STD bus reflects its origins by supporting the control lines necessary for this type of system. Most STD CPU cards which use Motorola or other processors supporting memory-mapped I/O have a mapping scheme which allocates some portion of the address space as I/O space (usually user-definable), and generates the appropriate control signals.

The STD bus backplane includes two separate power buses. The main power bus consists of pins 1 through 6. Pins 1 and 2 are five volt logic power, pins 3 and 4 are logic ground, and pins 5 and 6 are minus five volts. Pins five and six may alternately be used for a battery backup voltage and a power down indicator. The auxiliary power bus includes auxiliary postive on pin 55, auxiliary negative on pin 56, and auxiliary ground on pins 53 and 54. The auxiliary supply is usually

plus and minus 12 volts for RS232 drivers, or plus and minus 15 volts for analog circuits such as A/D convertors. Logic ground and auxiliary ground are separate circuits on the backplane, but may be connected on some cards or at the power supply.

Although the number of data and address lines available in a particular bus structure are important in determining the potential peak performance of a system using the bus, the unique characteristics of a bus which determine its "personality", or appropriateness for a particular application, are a result of the control lines which regulate the interactions among the various cards inserted in the backplane. The STD bus provides 22 control lines, and the underlying theme in their organization is simplicity. Other than the fact that components are physically arranged on different cards and a few more data and address buffers are present, there is little difference between many STD bus systems and a single board computer with similar capabilities. The advantage of the STD bus system is that you can mix and match I/O devices to suit your particular requirements. Following is a list of the control lines present on the STD backplane, along with a general description of their function. Note that signal names followed by a "*" are active low. This convention is usually used in STD bus documentation, and makes life easy for typesetters.

WR* - Write indicates that the current bus cycle is a data transfer from the current bus master to memory or an I/O port. Data should be latched into memory or the port on the trailing (rising) edge of WR*.

RD* - Read indicates that the current bus cycle is a data transfer from memory or an I/O port to the current bus master. Data should be latched on the trailing (rising) edge of RD*.

IORQ* - I/O request indicates that the current bus cycle is an access to an I/O port. For Z80 processors, an interrupt acknowledge cycle is indicated by IORQ* and STATUS1* being active simultaneously.

MEMRQ* - Memory request indicates the the current bus cycle is an access to memory (or memory-mapped I/O).

IOEXP - When high, I/O expansion indicates that the expansion page of I/O ports is being accessed.

MEMEX - When high, memory expansion indicates that the expanded memory area is being accessed. This signal is equivalent to A16, and is normally not used in systems with 20 bit multiplexed addressing.

REFRESH* - Refresh indicates that the current bus cycle is a refresh cycle for dynamic RAM. This signal is generated by the Z80 processor, but is seldom used in newer systems.

MCSYNC* - Machine cycle sync is a processor dependent signal that normally indicates the beginning of a machine cycle. On systems using extended addressing (more than sixteen address lines), this signal is used to latch the upper address lines.

STATUS1* - Status 1 is a signal from the current master to provide timing information for peripheral devices. If available from the current master, this signal should indicate an instruction fetch. On Z80 systems, this is the M1* signal which normal indicates an opcode fetch, but indicates an interrupt acknowledge cycle if it occurs together with IORQ*

.

STATUS0* - Status 0 is an additional timing signal for peripheral devices. Systems using Motorola processors usually put R/W* on this line. Unfortunately, Z80 systems typically don't use this line, and Intel systems put the equivalent signal on STATUS1* (opposite polarity no less). This is a major source of incompatability between various STD bus boards. (If you are currently using the STD bus and this is a problem for you, give me a call.)

BUSAK* - The bus acknowledge signal originates from the permanent master to indicate the bus is available to a temporary master.

BUSRQ* - The bus request signal is used by a temporary master such as a DMA controller to request control of the bus from the permanent master.

INTAK* - Interrupt acknowledge is generated by the permanent master to indicate the beginning of interrupt service. In vectored interrupt systems such as those using Z80 mode 2 interrupts, the interrupting device places the interrupt vector on the data bus in response to this signal. If more than one source of interrupts is present in a system using vectored interrupts, a hardware priority scheme is necessary to determine which interrupting device is being serviced.

INTRQ* - Interrupt request is used by peripheral devices to interrupt the permanent master.

WAITRQ* - Wait request is used to insert wait states in the current bus cycle. A valid address should be held on the bus when this signal is asserted.

NMIRQ* - Non-maskable interrupt is a high-priority interrupt request line.

SYSRESET* - System reset forces a hardware reset of all circuits.

PBRESET* - Push button reset is an input to the system reset circuitry which causes SYSRESET* to be asserted.

CLOCK* - This signal is the processor or system clock.

CNTRL* - Control is an auxiliary clock signal to provide additional timing information.

PCO - Priority chain out, along with PCI, is used to implement a position dependent daisy-chain priority scheme which may be used for arbitrating interrupt service or bus access. This active high signal indicates to lower priority cards that no higher priority card needs service.

PCI - Priority chain in may be used with PCO to arbitrate bus access or interrupt service. A card with PCI low will hold its PCO low and not respond to BUSAK* or INTAK*. A card requesting service with PCI high will hold its PCO low to suppress responses from lower priority cards, and respond to the appropriate acknowledge signal. A card not requesting service will pass the state of its PCI input to its PCO output.

The backplane serial priority chain using PCO and PCI is usually used for arbitrating interrupt priority. Parallel arbitration schemes for both bus and interrupt priority and serial schemes for bus priority may be implemented by interconnecting cards along the user edge. Although STD bus practice includes recommendations for such systems, they are very manufacturer dependent.

What are the advantages and disadvantages of the STD bus compared to other bus structures and single board computers ? Let's look at single board computers first. A single board computer will almost always be less expensive than an equivalent multi-board system. Aside from the cost of multiple PC boards, a multi-board system requires a backplane with expensive connectors for interconnecting system components, plus additional buffering to drive signals over the backplane. The physical dimensions of the two types of system are different, and one or the other may better meet the requirements of a particular design. A single board system has a flat form factor, while an STD system is shaped more like a brick or a shoebox. An STD system allows much greater flexibility in selecting I/O configurations than a single-board computer, and allows the addition of new I/O interfaces if system requirements change. Compared to other bus systems, the STD bus has the advantages of small card size, low cost, and probably the widest variety of interfaces from the largest number of manufacturers. In addition, the bus interface is very simple. Many systems with unique interfacing requirements may be put together with an off-the-shelf CPU, memory, and disk and terminal I/O, and an easily designed custom I/O board. This would be a more difficult undertaking on a more complex bus such as the VME bus. Other bus structures such as the VME bus support higher performance systems than the STD bus. For a multiuser system connected to disks, printers, and CRTs, the VME bus might be a better selection. One of the greatest disadvantages of the STD bus is that despite its name, there are several different varieties of STD bus which treat certain control signals differently. In practice however, the problems caused by these incompatabilities are most severe when integrating a CPU with dynamic memories and disk controllers. Usually an appropriate base configuration can be arrived at with a compatible CPU, memory, and disk controller. Then the user can pick and choose from the wide variety of I/O cards available on the STD bus.

The biggest attraction of the STD BUS is the wide variety of I/O cards which are available. Serial and parallel interfaces, disk controllers, SCSI interfaces, and LAN controllers in both conventional and intelligent configurations are supplied by numerous manufacturers. The STD Bus Buyer's Guide lists analog input or output products from 47 different manufacturers. A number of different types of display are supported, including video, LED, LCD, and plasma displays. As examples of some of the more unusual boards available, C-Matic Systems Ltd. supplies the STD 1920, a digital gauge head signal conditioning and interpolation board which provides a direct interface into Moire fringe type gauge heads, and the STD 1410 four channel LVTD/Inductive transducer interface with four independent channels of excitation and conditioning for interface to LVDT position guages. Contaq Technologies' UDM STD ultrasonic distance measuring board contains the complete signal processing necessary to perform non-contact distance measurement of objects between 0.5 and 35 feet, using ultrasonic techniques. Conway Engineering and Advanced Micro Systems both supply boards using National Semiconductor's voice synthesizer to provide voice output from the STD bus. Enlode Incorporated and Mimic, Inc. provide the reverse function with a cards that digitize audio inputs. Micro-Link and Parker Pneutronics supply STD bus cards with pneumatic valves or pressure switches and controls mounted directly on the card. If an application requires a special interface not available as an off the shelf product, prototyping cards are available which include the bus interface and an area for wire-wrapping. The interface to the bus is very simple and can be implemented with an address decoder, data bus buffer, and a few gates or a PAL(tm), allowing custom STD PC boards to be developed quickly and cheaply.

A number of Motorola family processors are available on the STD bus, ranging from the 6802 to the 68010. Along with proprietary monitors from various vendors, ROM and disk based systems are available using FORTH, PDOS, FLEX, OS9 Level 1 and 2, and OS9 68K. The most recent edition of the STD BUS Buyer's Guide from Control Engineering listed over 25 CPU boards using a Motorola family processor. Next month, I will briefly review some of the more interesting STD CPUs using Motorola processors. In the future I hope to discuss using the STD bus as an I/O expansion bus for other systems, including VME bus systems, systems based on the GMX Micro-20, and Apple Macintosh and Atari ST systems. Let me know if there is a particular topic which you would like to see addressed.

If you would like information about manufacturers of STD bus products or technical specifications for the various varieties of STD bus, you can contact the John Orr, Chairman of the STD Manufacturer's Group, c/o Pro-Log Corporation, 2560 Garden Road, Monterey, CA, 93940. The telephone number is 408 372-4593. The "STD Bus Buyer's Guide" is a good (but expensive - $44.95) reference which lists many STD bus products and manufacturers. Contact the "Control Engineering" Microcomputer Interface Group at 1-800-992-4447 (1-800-348-8342 in Illinois) if you are interested. And don't miss the ad for Bill West Incorporated elswhere in this issue !

Trademarks: OS9, Microware; Macintosh, Apple; ST, Atari.

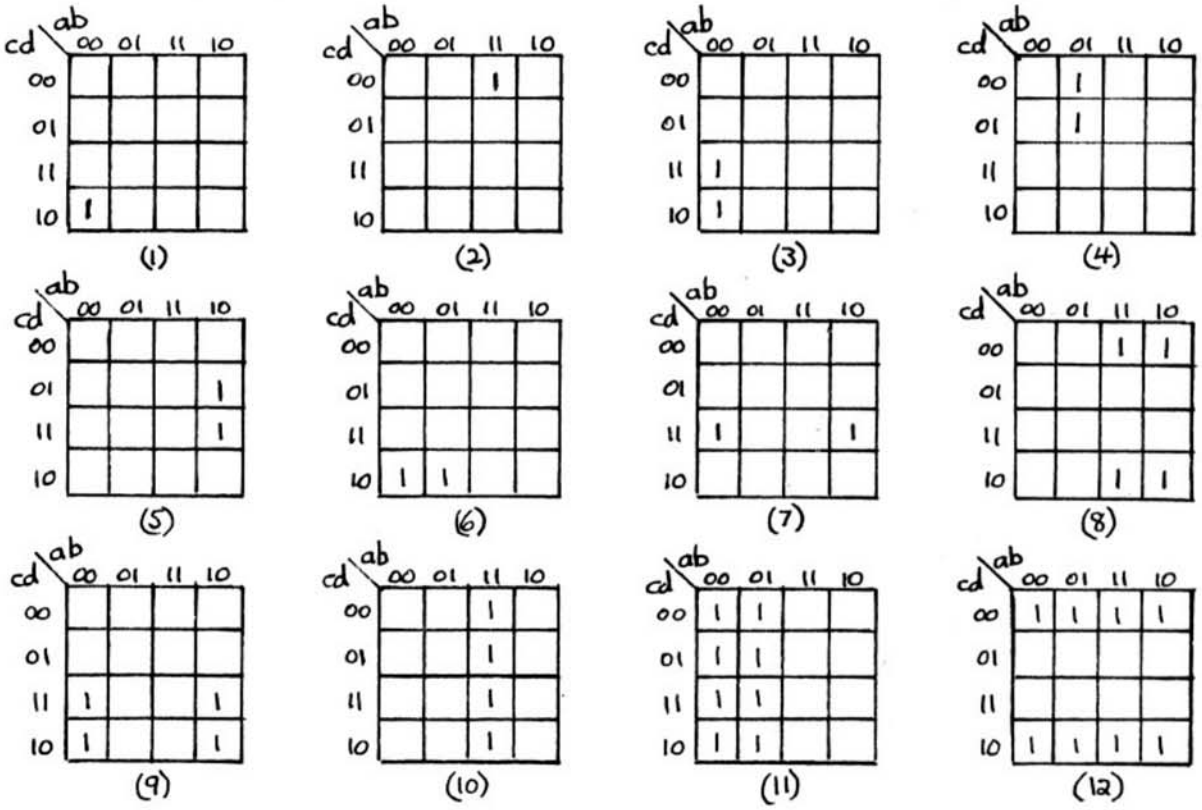
Logically Speaking

The following is the beginning of a continuing series. Most of you will remember Bob from his series of letters on XBASIC. If you like it or want more, let Bob or us know. We want to give you - *what you want!*

The Mathematical Design of Digital Control Circuits

By: R. Jones
Micronics Research Corp.
33383 Lynn Ave., Abbotsford, B.C.
Canada V2S 1E2

Solutions to TEST THREE



Solutions to TEST FOUR

```
(i)   bd' + b'd              (b' + d') (b + d)
(ii)  a'c' + bd'             (a' + b) (a' + d') (b + c') (c' + d')
(iii) a' + d                 a' + d
(iv)  b' + d                 b' + d
(v)   b'c + bc' + a'c'd'          (b' + c') (a' + b + c) (b + c + d')
   OR b'c + bc' + a'b'd'
(vi)  ad' + bd' + abc'            (a + b) (a + d') (b + d') (c' + d')
(vii) b' + d'                b' + d'
(viii) abc + bcd + ad + bc
       (a' + b + c) (a' + b' + c') (a' + b' + d) (b' + c' + d) (b + c + d)
   OR  (a' + b + c) (a' + b' + c') (a' + c + d) (b' + c' + d) (b + c + d)
```

Mile 3 - heading for Mile 4

IMPLICANTS

Here are definitions for a few words you're likely to come across in literature on this subject, but you need make no special effort to remember them right now.

IMPLICANT Every minterm on a K-map is an implicant. That is, each single "1". So also is ANY loop that you can make on the map, whether it's the largest possible to include a particular minterm, or whether it's not.

PRIME IMPLICANT Any loop on the map, AS LONG AS IT'S THE LARGEST POSSIBLE LOOP to include a particular minterm.

ESSENTIAL PRIME IMPLICANT A prime implicant which MUST be retained in the reading of the K-map. In Map (v) of TEST FOUR, for instance, the top left-hand "1" may be looped either with the "1" to the right, OR with the "1" in the bottom left-hand corner. Here, where you have a choice, neither is ESSENTIAL. The other two loops are, however.

THE SYNTHESIS OF COMBINATIONAL CONTROL-CIRCUITS

Here's what you've all been waiting for! Designing an actual circuit, albeit a simple one. I think the best way to demonstrate the technique is to work on a hypothetical set of specifications, so we'll assume that a customer wishes us to construct a circuit to initiate the automatic packaging of electrical switch-boxes. Let us therefore imagine a conveyor-system carrying streams of switch-boxes towards the packaging-machine, and just before the machine we need a little switch-panel, with a small bank of switches which can be toggled ON or OFF by a human operator, who scans the switch-boxes as they approach him on the belt. These switches are labelled 'FUSED', 'CHROMED', etc, as we'll determine from the supplied set of specs. Here they are :

1. The switch will be packaged if it's seam-welded and painted grey, OR if the case is unpainted but chrome-plated.

2. The switch will NOT be packaged if fuses are incorporated and the case is seam-welded and chrome-plated. It will also NOT be packaged if the case is spot-welded and painted.

3. If fuses are installed and the case is painted, it may or may not be packaged, at OUR discretion - whichever leads to the simpler circuit.

Now we can imagine the operator flipping the bank of control-switches in accordance with the type of switch-box approaching, which in turn will either divert the switch-box off to one side, OR allow it to proceed into the packaging machine. It's not likely that you'll be called on to design this very circuit, but you'll certainly use exactly the technique which I'm about to describe.

Our first step, as in the case of Club membership, is to pick out the variables involved and assign letters to them, as set out below :

```
w = seam-welded          w' = spot-welded
p = painted              p' = chromed (ie, unpainted)
g = grey                 g' = not grey
f = fused                f' = unfused

T (the transmission function) = 1 for package
                              = 0 for don't package.
```

And just to make it a little more interesting, we'll throw in our own observation that "chromed AND grey" is an impossible condition, which we'll call Specification 4.

Our second step is to reduce these specs to algebraic form :

```
1.  If wpg + p'    then T = 1
2.  If fwp' + w'p  then T = 0
3.  If fp          then T = ø
4.     p'g               T = ø
```

Both these symbols, Ø and ø, are pronounced "fy", as in "magnify", and are upper and lower-case Greek letter "phi" respectively. The first, Ø, will be used whenever we have a choice, or don't care to make a decision for the moment. For this reason, it's often referred to as a "don't care". The second, ø, will be used to designate an impossible condition. As far as I know, no such distinction is made by ANY OTHER design-system, and you'll find that other authors use only the one symbol, Ø, for both conditions. However, in my experience, two separate symbols DO have certain distinct advantages, as we'll see shortly.

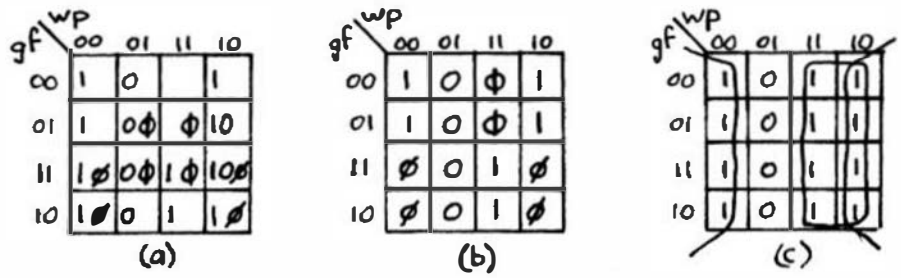


Diagram 10

The third step is to enter these expressions on a "synthesis map", which is no more than a K-map, but with the difference that multiple symbols may be entered in its squares, provided that no more than one of each kind is so entered. Diagram 10a shows our first attempt at entering the appropriate symbol (1, 0, Ø or ø) according to the terms in the four expressions obtained from the specifications. If the specs were all in order we should see a completely-filled map with exactly one symbol in each square, and the fact that this is NOT so indicates that we have logical problems in our specs.

CONTRADICTIONS AND AMBIGUITIES

Different entries in one square indicate a "contradiction", while an empty square arises from an "ambiguity", or unspecified condition.

For example, square 1001 indicates that the customer wants to both package and NOT package under the condition wp'g'f. Square 1011, wp'gf, is even worse - not only is there a direct contradiction in the specs, but this is a situation which is impossible to occur anyway.

On the other hand, the empty square 1100, wpg'f', needs an instruction from the specifier. So let's assume that when we contact him and point all this out, he says that he doesn't care what we do with 1100, so we insert a Ø in this square. We'll further assume that he decides that any 1,0 contradictions should be resolved in favour of 1.

Any squares which contain an "impossible" ø are, of course, immediately converted to a definite ø, as any instruction other than this is meaningless when the situation is impossible. Squares containing a "don't care" Ø, however, must give precedence to a definite 1, or 0, occupying the same square as itself. The customer has already told us to resolve 1,0 contradictions in favour of 1, so we'll not have any 1,0s in the same square any longer.

Applying all this to K-map 10a results in K-map 10b, which is now an acceptable map, as it contains EXACTLY one entry per square. We find that if we read out the 1s on this map we obtain the control expression :

p'g' + wpg

with no mention of "f", so obviously the question of whether the switch-box is fused or not is totally irrelevant. Note in passing that again ONCE WE HAVE OUR PROBLEM SAFELY COMMITTED TO A MAP, THE PROBLEM IS AUTOMATICALLY SOLVED.

BUT, and this is EXTREMELY important in designing good reliable circuits, WE CAN MAKE USE OF THE PHI's IN BOTH FORMS TO MINIMISE OUR NETWORK STILL FURTHER.

CIRCUIT MINIMISATION USING PHI

We can use the Øs (after all, the customer has said that he doesn't care what happens under these conditions) as if they were 1s, thus reducing our readout of "wpg" to "wp", to give a control-expression of

p'g' + wp

Moreover, if a particular set of conditions can NEVER arise, we COULD design our circuit to hand out free drinks to everyone under these circumstances BECAUSE THIS WILL NEVER BE CALLED ON TO OCCUR. However, as our only aim is to obtain nice large loops on our K-map, we'll settle for making the øs into 1s as well. We thus end up with Diagram 10c above - shown for demonstration purposes only, as we don't ACTUALLY convert the phi's to 1s; we simply read them as if they were. Accordingly we end up with a reading of :

p' + w

by looping together the first and last columns to give p', and the last two columns to give w. By making optimum use of the phi's we have not only reduced the total number of contacts from an initial 5 to a final 2, but have also eliminated another variable so that we now only need 2 switches to be set by the operator, instead of our earlier 3. This helps to reduce the possibility of human error in setting the switches, especially at the end of a long, tiring shift, as he now only has to determine the type of welding on the switch-boxes, and whether they're chromed or not. Our circuit can be formed directly from the switches themselves, without the need of any relays, as shown in Diagram 11 :

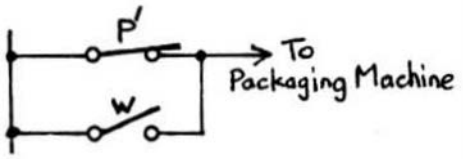


Diagram 11

READING PHI TO FORM OPTIMUM LOOPS

Don't be misled into thinking that ALL the phi's MUST be completely used up. We can pick and choose, and select only those which are necessary to give us the best reading of the K-map formed from the synthesis map, the remainder being interpreted as 0s.

In the case of this circuit, reading the 0s, and then complementing, results in exactly the same circuit, indicating that there is no alternative form. But keep in mind - THIS IS IMPORTANT - when reading 0s, that you can now make a completely different decision regarding the phi's. Just because a 1s reading leaves us with a certain collection of "phi-0s" doesn't mean that we're stuck with this interpretation. No, this is a completely separate and independent read-out of the map! Sometimes it happens that the phi's are more advantageously distributed for a 0s read-out, and we can actually end up with a smaller circuit this way. Sometimes, of course, it's a more complicated circuit! So, to be absolutely certain, we should do both types of map-reading.

A BRIEF RE-CAP

Before trying a second circuit-design, let's just take a final look at this whole process, and observe how difficult (nay, impossible) it would be to discover the logical contradictions in our initial set of written specs, or that there's one combination of switch-box characteristics that's not even covered. Even though this is a VERY small set of specs! It's difficult enough to determine this EVEN WHEN WE KNOW FROM OUR SYNTHESIS-MAP that these logical problems exist. This is what makes it so hard sometimes to design a circuit by the old "seat-of-the-pants" method, where a designer simply sits down and begins drawing up a circuit directly from the specs. In addition, very often he has so much trouble trying to design for what APPEARS to be "possible", according to the specs, that he sees no need at all to concern himself with the added burden of what is "impossible" in a circuit. There must be thousands of designers who don't realise what a terrific advantage both types of phi can offer. In fact, the more of them there are, the more flexibility we have in choosing our final network, and, in general, the more compact the final design is going to be! So let's keep an eye open for these little fellows, and use them to our advantage at every opportunity. By the way, the symbol Ø was originally chosen because it actually looks like a 1 and a 0 superimposed, meaning that, for the moment, we haven't decided which it's going to be.

ANOTHER COMBINATIONAL CIRCUIT DESIGN

To re-inforce our new technique, let's assume that we have a mechanical press, which has to cycle automatically if there's a component positioned correctly AND the die is up to temperature, OR if the Start-Switch is operated (possibly to test for smooth operation by a set-up technician). It should stop if the Stop-Switch is operated.

That certainly seems straightforward enough - no logical problems there that I can determine right now! How about you? Do you see any?

Let's design two different machines. One where the Stop and Start switches are interlocked to prevent both of them from being actuated simultaneously (that seems a reasonable precaution), and two, where these switches are interlocked to prevent both ON-conditions from occurring simultaneously, and also both OFF-conditions (that is, if one is ON, the other must be OFF). If we call the Start-Switch X1 and the Stop-Switch X2, this means that in Circuit-1 X1.X2 = ø, and in Circuit-2 X1.X2 and X1'.X2' = ø.

OK, let's allocate our variables, and set out our various transmission functions for these circuits.

```
Let  a = Part in position            b = die up to temperature
     c = STOP-Switch operated        d = START-Switch operated

then    ab + d      T = 1
           c        T = 0
          cd        T = ø  (circuit 1)
 OR cd + c'd'       T = ø  (circuit 2)
```

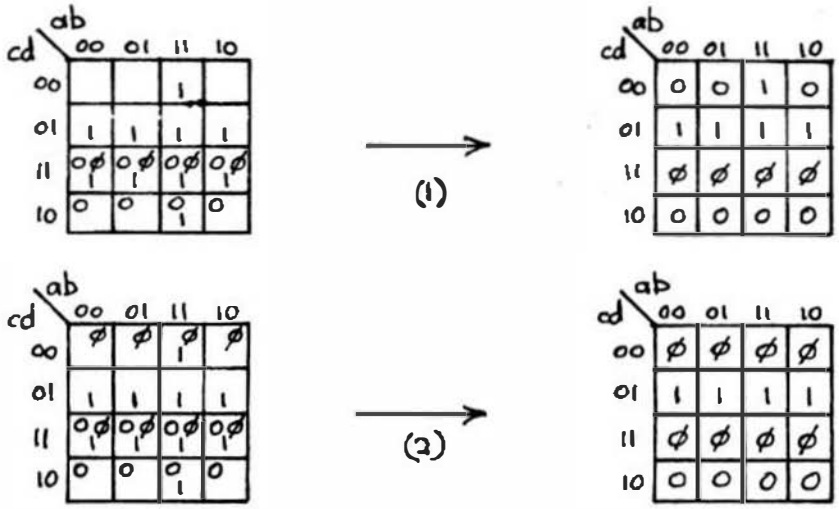


Diagram 12

Observe the triple-contradiction along the whole of the 11-row of the upper synthesis-map. In this situation, we don't really need a decision from the customer, because this row is an "impossible" row. On the othr hand, if these were "don't care" phi's, we would HAVE to go back to the customer and explain that in effect he's telling us (i) that the press MUST cycle under these conditions, (ii) that it must NOT cycle and (iii) that he doesn't care what it does. As it's an "impossible" situation, however, we merely advise the customer of the flaw in his specs (nicely, of course), and eliminate both his 1s and his 0s in favour of ø in this row. Now you see why I developed the idea of having two distinct phi's!! In addition there is a simple 1,0 contradiction in square 1110 (interpretation - part in position, die up to temperature, STOP-switch pressed, START-switch not pressed), which common-sense dictates should be resolved in favour of 0.

Obviously, even though the press conditions are OK, if someone is activating the STOP-switch the press MUST stop. Maybe someone has his fingers or his head caught in the press, and must be released! There are also a few ambiguities, or unspecified conditions, in the top row, which a further application of common-sense will normally resolve in favour of 0. This produces the K-map to the right, from which we read out abc' + d.

The lower synthesis-map gives us an additional row of øs in the top row, which we resolve completely in favour of ø, resulting in the K-map to the right. we can read this as either c' (by looping the top 2 rows) or as d (by looping the centre 2 rows). As this circuit does not take into account ANY of the internal press conditions, we would be well advised to go back to our customer to verify whether this is what he REALLY desires!

Isn't it amazing how these little 1s and 0s and phi's show up the logical flaws in specifications, which might otherwise go undetected!! The synthesis-map also highlights ambiguities, so that we can be ABSOLUTELY certain that we've not overlooked a particular combination of circumstances in our design, leaving the machine or circuit undecided what to do should this combination occur.

Now it's your turn to try your hand. Don't concern yourself with whether these are realistic machines or not. After all, if this is what the customer wants, who are we to argue with him? Don't forget, we're in this business to earn some money with our fabulous designs! So let's go make our fortune!! Here then is

TEST FIVE

1. This may sound a little confusing, so take it one step at a time. Assume we have an automatic car-washing establishment, whose entrance-barrier will be raised if the right coins are dropped into a collection-box. The presence of a car is indicated by a detector-pad in the entrance-way, and the cars are counted by a photocell device.

The proprietor requires a light in his office to be ON if the barrier is down and the photocell lit and no car present, OR if a car is present, the light-source OK and the photocell not lit (in other words, everything is functioning normally). On the other hand, the office-light is to go out if the light-source for the photocell goes out, OR if the barrier is raised when no car is present (but the light-source is OK and the photocell is lit). Design the circuit to control the office-light.

HINT : It's impossible, with no car present, for the light-source to be OK and the photocell not lit, or vice-versa.

2. Design a release-mechanism for a chocolate-bar machine, such that the pull-drawer can be operated if '2 dimes plus 1 nickel' OR 1 quarter is inserted, and then a button pressed. The release mechanism must not operate if there are no chocolate-bars in the machine. (Let's not worry about returning the customer's money!) Further, both sets of coins cannot be inserted simultaneously.

That should keep you all quiet for a little while! Our journey will get more and more interesting from now on in, and it will be a long while before we come across anything as strange as the Laws of Boolean swamp. Soon, we'll even be leaving K-maps behind, and playing with decimal numbers instead, as I promised earlier. So next time, we'll move out of Combinational-Circuit country, and into the land of Sequential-Circuits. An easy transition, I can assure you. Well, anyway, it won't be a hard one!

..... End of Mile 3. OK, have it your way -- Start of Mile 4!

EOF

Converting the IBM-XT Keyboard to Replace the CT-82

s/Clive R. Allan
23 Henwood Street
Blackburn South Victoria 3130
Australia

Here is a neat little hardware project using the MC68705P3 8-bit Eprom microcomputer that you and your readers may find interesting.

The design revolves around converting an IBM-XT type keyboard serial output into 7 bit parallel ascii with strobe line. In my particular case I wanted it to emulate the SWTPC's CT-82 keyboard which meant some additional circuitry.

Before getting into the nitty gritty I would like to point out that the original idea, software and hardware development was done by 2 fellow micro club members, Frank Criveili and Wayne Fordham. I owe them much in helping me fine tune this project to get it to work on the XT keyboard.

In wanting to replace my rapidly decaying keyyyboooard I looked around fr a suitable alternative. Unfortunately the peple at SWTPC developed an interesting way of getting both standard parallel ascii and parallel cursor control output. They split the 2 into affectively two totally seperate keyboards, each with its own data and strobe lines. Very good you might say, but rather difficult to replace.

This was when I discovered that my 2 friends had developed a way of converting an IBM keyboard to 7 bit ascii parallel output. I duly went out and purchased an IBM keyboard clone, borrowed one of thier MC68705's and proptly found it did'nt work at all. So much for doing it the easy way!

I soon discovered that not all IBM keyboards are the same. They all have the same hardware requirements, but thats where the simularity ends. From what I can gather there are at least three different units. 1) PC keyboard - Earliest keyboard has a 10 bit data stream 2) XT keyboard - Cheaple IBM has 9 bit data stream 3) AT keyboard - Not the same as above (haven't worked it out)

The keyboard I purchased was a switchable XT/AT modl and I used it in the XT mode to get the little indicator lights to work (caps lock etc). There are 4 connections to the keyboard; power (5v), ground, serial data output and serial clock output.

In operation, when a key is depressed 9 clock pulses are sent out on the serial clock line (approx 50 micro Secs apart) and simultaneously data is sent out over the data line. By sampling the condition of the data line at every clock pulse the data can be determined. If the first clock pulse and associatd data is ignored, the following 8 make up a unique 8 bit code for that particular key. B0 is sent out first, B7 last. B7 is special in that it indicates whether the key was depressed (B7=0) or the key was released (B7=1). Once the data is in, it is only a matter of code conversion and in my particular case directing to keyboard output or cursor output.

I used the INT input of the MC68705 to receive the serial clock from the keyboard. This is done in an interupt mode because the very short duratio of the clock pulse (<5 micro Secs) prohibited reliable level checking of a port pin using polling. The INT can be made to bring up an interupt much like a CA pin of a 6821.

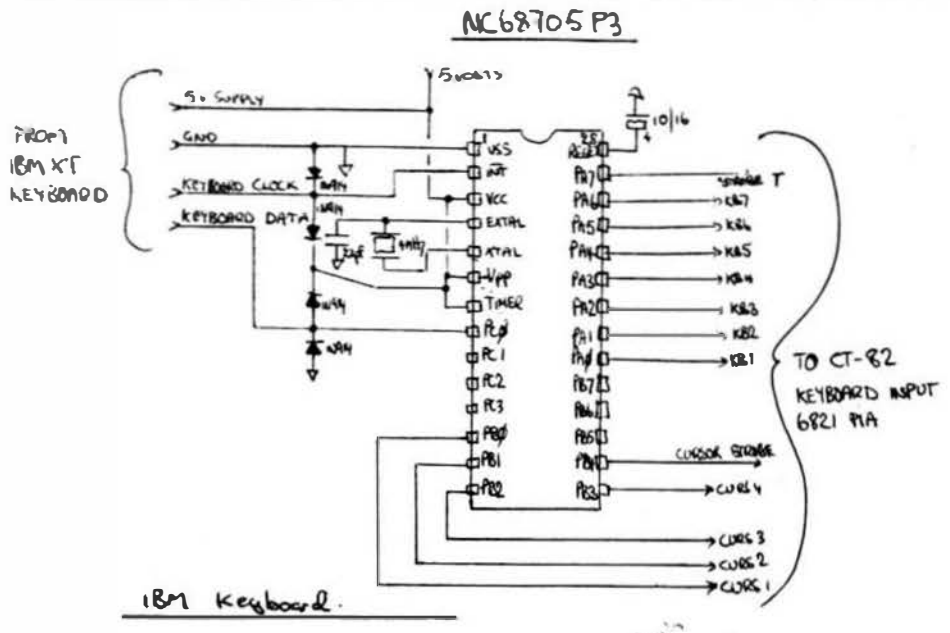
Serial data is shifted into temp store and a counter is incremented. The main program checks this count and when it has reached 9 (because I do not want the first bit) a valid byte is in the temp store. You could change this count to suit say a PC keyboard in which case it would count to 10 before flagging a valid byte.

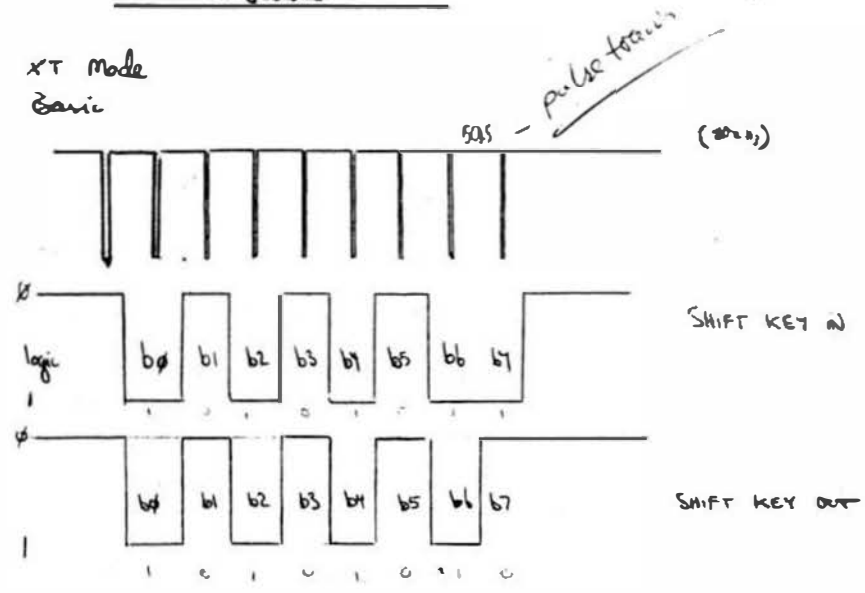
The main program is fairly straight forward in converting this byte into its ascii equivalent. The program also remembers if nums lock, caps lock, shift etc is active. Remember, as far as the keyboard is concerned all of hese keys (Alt, Caps Lock, Num lok etc) are just that - keys, they d not modify the output of other keys. It is up to the 6805 to track of their condition. E.g don't unplug the keyboard then plug it back in and expect it to work properly as the keyboard's condition may not be what the 6805 thinks it should be.

One last point that should be made and some of you may have already figured it out, is that because of the way of collecting data from the keyboard there is a risk that a serial clock pulse is missed or more likely a glitc is mistaken for a clock pulse the 6805 is permanently out of sync with the keyboard and the only way to correct is to reset the cpu.

I found this out the hard way when I flicked the switch on the keyboard to AT and back to XT then wondered why it stopped working. Fortunately there is an inbuilt timer in the cpu. All I did was to work out how long valid data took to enter and be processed. If this value is blown the timer initiates an Interupt whose routine clears down all the flags affectively eseting the cpu.

```
*****************************************************
*                                                   *
*        I B M - X T       K E Y B O A R D          *
*                  D E C O D E R                    *
*                                                   *
*****************************************************
*                                                   *
*  IBM-XT KEYBOARD CONVERSION TO ASCII PARRALLEL    *
*  TO BE USED AS REPLACEMENT KEYBOARD FOR CT-82      *
*                                                   *
*  Original idea, software and hardware develop-    *
*  ment by     Frank Crivelli & Wayne Fordham        *
*                                                   *
*  Modified to suit XT keyboard and CT-82 by:       *
*  Clive Allan                                      *
*  23 Henwood Street, Blackburn South 3130          *
*  Victoria, Australia.                             *
*                                                   *
*  Date 12th May 1987                               *
*                                                   *
*****************************************************
*                                                   *
*  This Software routine has to be assembled by a   *
*  6805 assembler; then to be transferred to a      *
*  MC68705P3 micro.                                 *
*                                                   *
*  HARDWARE      PORT-A = BITS 0-6 KEYBOARD ASCII   *
*                         OUTPUT TO TERMINAL.       *
*                         BIT 7 KEYBOARD STROBE     *
*                                                   *
*                PORT-B = BITS 0-3 CURSOR CONTROL   *
*                         OUTPUT TO TERMINAL        *
*                         BIT 4 CURSOR STROBE       *
*                                                   *
*                PORT-C = BIT 0 SERIAL DATA FROM    *
*                         FROM XT KEYBOARD          *
*                                                   *
*                INT PIN (2) SERIAL CLOCK FROM XT   *
*                KEYBOARD                           *
*                                                   *
*****************************************************
*
*  GLOBAL EQUATES
*
0000  porta   equ   $0
0001  portb   equ   $1
0002  portc   equ   $2
0004  ddra    equ   $4          porta ddr
0005  ddrb    equ   $5          portb ddr
0006  ddrc    equ   $6          portc ddr
0008  tdr     equ   $8          timer data reg
0009  tcr     equ   $9          timer control reg
```





- key in b7 = Ø key out b7 = 1 . Above example should show $D5 in - $55 out

```
                *
                *  RAM VARIABLES
                *

0010                    org     $010

0010            kstate  rmb     1               status of function keys

                *         BIT     COMMENT
                *          0      Shift key operated
                *          1      Caps key lock activated
                *          2      Control key active
                *          3      Numbers lock activated
                *          4      Alt key operated
                *          5-7    Not used

                * Temp storage for incoming data

0011            store   rmb     1               raw data
0012            count   rmb     1               counter for incoming data
0013            data    rmb     1               processed binary data

                *
                *  MAIN PROGRAM STARTS HERE
                *

0100                    org     $100

                * Initialize routine to set paramenters

0100 3F 10      reset   clr     kstate          clear variables
0102 3F 11              clr     store
0104 3F 12              clr     count
0106 3F 13              clr     data

0108 A6 FF              lda     #$FF            set ddr for outputs
010A B7 04              sta     ddra            on ports A & B
010C B7 05              sta     ddrb
010E B7 08              sta     tdr             reset watch-dog timer
0110 A6 FE              lda     #$FE            set port-C bit0 input
0112 B7 06              sta     ddrc
0114 1E 00              bset    7,porta set     key/brd &
0116 18 01              bset    4,portb Cursor  strobe lines highb
0118 A6 07              lda     #%00000111      set timer/128 clk
011A B7 09              sta     tcr             inhibite timer interupt
011C 9C                 rsp                     reset stack
011D 9A                 cli                     clear interupt mask
>011E CC 0151           jmp     readml          go to primary

                ****************************************
                * INTERUPT ROUTINE                     *
                ****************************************
                * THIS ROUTINE IS CALLED EVERY TIME THE
                * IBM-XT KEYBOARD CLOCK LINE GOES LOW.
                * IT WILL BE CALLED 9 TIMES TO STORE
                * ONE WORD, 50 micro SECS BETWEEN CALLS.
                *****

0121 9B         go      sei                     mask further interupts
0122 B6 02              lda     portc           grab k/b data line
0124 A4 01              anda    #$01
0126 38 11              lsl     store           shift data in
0128 BB 11              adda    store
012A B7 11              sta     store
012C 3C 12              inc     count           inc counter
012E 9A                 cli                     clear int mask
012F 80                 rti

                ****************************************
                * TIMER WATCH-DOG ROUTINE
                *
                * ALL 8 BITS HAVE NOT BEEN ENTERED IN 30 mSec
                * TIME COUNTER HAS FALIEN THROUGH. RESET
                * FLAG VARIABLES.
                *
0130 3F 11      time    clr     store
0132 3F 13              clr     data
0134 3F 12              clr     count
0136 A6 FF              lda     #$FF
0138 B7 08              sta     tdr             reset timer
013A 1F 09              bclr    7,tcr clear     to allow int
013C 80                 rti
                ****************************************
                *
                * OUTPUT ROUTINES
                *
                * OUTPUT TO KEYBOARD AND STROBE
                * DATA HELD IN ACC-A
                *
013D AA 80      putkey  ora     #$80            don't alter strobe
013F B7 00              sta     porta           output data
0141 1F 00              bclr    7,porta         strobe low
0143 1E 00              bset    7,porta         reset strobe
0145 20 0A              bra     readml          return to start

                *
                * OUTPUT TO CURSOR AND STROBE
                * DATA HELD IN ACC-A
                *
0147 AA 10      putchr  ora     #$10            don't alter strobe
0149 B7 01              sta     portb           output data
014B 19 01              bclr    4,portb         strobe low
014D 18 01              bset    4,portb         reset strobe
014F 20 00              bra     readml          return to start

                *
                * PRIMARY PROGRAM STARTS HERE; IS ONLY
                * ACTIVE AFTER SERIAL BIT STREAM HAS
                * FINISHED FROM KEYBOARD
                *
0151 A6 FF      readml  lda     #$FF            reset watchdog
0153 B7 08              sta     tdr
0155 B6 12      read3   lda     count           check if valid data
0157 A1 09              cmpa    #$9
0159 27 06              beq     out             go process data
```

```
015B A1 00              cmpa    #$00                is it idle?
015D 27 F2              beq     readm1              yes, restore watch-dog
015F 20 F4              bra     read3               no, could be stuck wait

                * Data is in but order reversed, clean up.

0161 A6 08      out     lda     #$8
0163 39 11      movel   rol     store               get data out
0165 36 13              ror     data                stuff it in
0167 4A                 deca
0168 26 F9              bne     movel
016A 3F 12              clr     count               reset bit counter
016C 3F 11              clr     store               clear data catch; ready for more

                * CHARACTER NOW IN TEMP STRORE CALLED 'DATA'
                * CHECK CONDITION OF FUNCTION KEYS AND CONVERT
                * APPROPRIATE COMMAND TO ASCII PARRALLEL.
                *
                * KEYBOARD DATA FORMAT: 8 BITS DATA (7TH BIT
                * 0 - KEY PRESSED, 1 - KEY RELEASED.
                *
                * bin code:Function bin code:Function
                *
                *   01      ESC       0F      I<<<
                *   02        1       10      Q
                *   03        2       11      W
                *   04        3       12      E
                *   05        4       13      R
                *   06        5       14      T
                *   07        6       15      Y
                *   08        7       16      U
                *   09        8       17      I
                *   0A        9       18      O
                *   0B        0       19      P
                *   0C        -       1A      {
                *   0D        =       1B      }
                *   0E       BS       1C      CR
                *
                *   1D       CTRL     1E      A
                *   1F        S       20      D
                *   21        F       22      G
                *   23        H       24      J
                *   25        K       26      L
                *   27        ;       28      :
                *   29        '
                *
                *   2A      SFT (L)   3A      CAPS LK
                *   2B        \       3B      F1
                *   2C        Z       3C      F2
                *   2D        X       3D      F3
                *   2E        C       3E      F4
                *   2F        V       3F      F5
                *   30        B       40      F6
                *   31        N       41      F7
                *   32        M       42      F8
                *   33        ,       43      F9
                *   34        .       44      F10
                *   35        /       45      NUM LK
                *   36      SFT (R)   46      SRL LK
                *   37        *       47      HOME
                *   38      ALT       48      ^
                *   39      SPACE     49      PG UP
                *
                *   4A        _       4B      <--
                *   4C                4D       -->
                *   4E        +       4F      END
                *   50        v       51      PG DN
                *   52      INS       53      DEL
                *   54      SYS RES

                * FIRST CHECK FOR FUNCTION KEY

016E B6 13              lda     data                is key being released?
0170 2B 3D              bmi     relcod              bit 7 set
0172 A1 10              cmpa    #$1D                is it control key?
0174 27 16              beq     mctrls              yes, set bit2 kstate
0176 A1 2A              cmpa    #$2A                is key left shift key
0178 27 16              beq     mshift              set bit 0 in kstate
017A A1 36              cmpa    #$36                is key right shift key
017C 27 12              beq     mshift
017E A1 3A              cmpa    #$3A                is key caps lock key
0180 27 12              beq     mcapsl              set bit 1 in kstate
0182 A1 38              cmpa    #$38                is key alt key
0184 27 19              beq     maltss              set bit 4 kstate
0186 A1 45              cmpa    #$45                is key num lock
0188 27 19              beq     mnumsl              set bit 3 in kstate
018A 20 3F              bra     sortc               none of the above.

                * TOGGLE FLAGS ACCORDING TO KEYS
                * SHIFT, ALT, CTRL SET AND RELEASE
                * NUMS LOCK, CAP LOCKS.

018C 14 10      mctrls  bset    2,kstate
018E 20 1C              bra     readm2
0190 10 10      mshift  bset    0,kstate
0192 20 18              bra     readm2
0194 02 10 04   mcapsl  brset   1,kstate,mcaplo
0197 12 10              bset    1,kstate
0199 20 11              bra     readm2
019B 13 10      mcaplo  bclr    1,kstate
019D 20 0D              bra     readm2
019F 18 10      maltss  bset    4,kstate
01A1 20 09              bra     readm2
01A3 06 10 04   mnumsl  brset   3,kstate,mnumlo
01A6 16 10              bset    3,kstate
01A8 20 02              bra     readm2
01AA 17 10      mnumlo  bclr    3,kstate
>01AC CC 0151   readm2  jmp     readm1              finish with data
```

FORTHBUILDER is a stand-alone target compiler (crosscompiler) for producing custom Forth systems and application programs. All of the 83-standard defining words and control structures are recognized by FORTHBUILDER.

FORTHBUILDER is designed to behave as much as possible like a resident Forth interpreter/compiler, so that most of the established techniques for writing Forth code can be used without change.

Like compilers for other languages, FORTHBUILDER can operate in "batch mode".

The compiler recognizes and emulates target names defined by CONSTANT or VARIABLE and is readily extended with "compile-time" definitions to emulate specific target words.

FORTHBUILDER is supplied as an executable command file configured for a specific host system and target processor. Object code produced from the accompanying model source code is royalty-free to licensed users.

F, CCF, S - $99.95

DATABASE ACCOUNTING

XDMS from Westchester Applied Business Systems

FOR 6809 FLEX-SK*DOS(5/8")

Up to 32 groups/fields per record! Up to 12 character filed name! Up to 1024 byte records! User defined screen and print control! Process files! Form files! Conditional execution! Process chaining! Upward/Downward file linking! File joining! Random file virtual paging! Built in utilities! Built in text line editor! Fully session oriented! Enhanced forms! Boldface, Double width, Italics and Underline supported! Written in compact structured assembler! Integrated for FAST execution!

XDMS-IV Data Management System

XDMS-IV is a brand new approach to data management. It not only permits users to describe, enter and retrieve data, but also to process entire files producing customized reports, screen displays and file output. Processing can consist of any of a set of standard high level functions including record and field selection, sorting and aggregation, lookups in other files, special processing of record subsets, custom report formatting, totaling and subtotaling, and presentation of up to three related files as a "database" or user defined output reports.

POWERFUL COMMANDS!

XDMS-IV combines the functionality of many popular DBMS software systems with a new easy to use command set into a single integrated package. We've included many new features and commands including a set of general file utilities. The processing commands are Input-Process-Output (IPO) oriente which allows almost instant implementation of a process design.

SESSION ORIENTED!

XDMS-IV is session oriented. Enter "XDMS" and you are in instant command of all the features. No more waiting for a command to load in from disk! Many commands are immediate, such as CREATE (file definition), UPDATE (file editor), PURGE and DELETE (utilities). Others are process commands which are used to create a user process which is executed with a RUN command. Either may be entered into a "process" file which is executed by an EXECUTE statement. Processes may execute other processes, or themselves, either conditionally or unconditionally. Menus and screen prompts are easily coded, and entire user applications can be run without ever leaving XDMS-IV

IT'S EASY TO USE!

XDMS-IV keeps data management simple! Rather than design a complex DBMS which hides the true nature of the data, we kept XDMS-IV file oriented. The user view of data relationships is presented in reports and screen output, while the actual data resides in easy to maintain files. This aspect permits customized presentation and reports without complex redefinition of the database files and structure. XDMS-IV may be used for a wide range of applications from simple record management systems (addresses, inventory ...) to integrated database systems (order entry, accounting...)

The possibilities are unlimited...

FOR 6809 FLEX-SK*DOS(5/8") $249.95

ASSEMBLERS

ASTRUK09 from S.E. Media -- A "Structured Assembler for the 6809" which requires the TSC Macro Assembler.

F, S, CCF - $99.95

Macro Assembler for TSC -- The FLEX, SK*DOS STANDARD Assembler.

Special -- CCF $35.00; F, S $50.00

OSM Extended 6809 Macro Assembler from Lloyd I/O. -- Provides local labels, Motorola S-records, and Intel Hex records; XREF. Generate OS-9 Memory modules under FLEX, SK*DOS.

*FLEX, SK*DOS, CCF, OS-9 $99.00*

Relocating Assembler/Linking Loader from TSC. -- Use with many of the C and Pascal Compilers.

F, S, CCF $150.00

MACE, by Graham Trott from Windrush Micro Systems -- Co-Resident Editor and Assembler; fast interactive A.L. Programming for small to medium-sized Programs.

F, S, CCF - $75.00

XMACE -- MACE w/Cross Assembler for 6800/1/2/3/8

F, S, CCF - $98.00

UTILITIES

Basic09 XRef from S.E. Media -- This Basic09 Cross Reference Utility is a Basic09 Program which will produce a "pretty printed" listing with each line numbered, followed by a complete cross referenced listing of all variables, external procedures, and line numbers called. Also includes a Program List Utility which outputs a fast "pretty printed" listing with line numbers. Requires Basic09 or RunB.

O & CCO obj. only -- $39.95; w/ Source - $79.95

BTree Routines - Complete set of routines to allow simple implementation of keyed files - *for your programs* - running under Basic09. A real time saver and should be a part of every serious programmers tool-box.

O & CCO obj. only - $89.95

Lucidata PASCAL UTILITIES (Requires Pascal ver 3)

XREF -- produce a Cross Reference Listing of any text; oriented to Pascal Source.

INCLUDE -- Include other Files in a Source Text, including Binary - unlimited nesting.

PROFILER -- provides an Indented, Numbered, "Structogram" of a Pascal Source Text File; view the overall structure of large programs, program integrity, etc. Supplied in Pascal Source Code; requires compilation.

F, S, CCF --- EACH 5" - $40.00, 8" - $50.00

DUB from S.B. Media -- A UniFLEX BASIC decompiler Re-Create a Source Listing from UniFLEX Compiled basic Programs. Works w/ ALL Versions of 6809 UniFLEX basic.

U - $219.95

LOW COST PROGRAM KITS from Southeast Media The following kits are available for FLEX, SK*DOS on either 5" or 8" Disk.

1. BASIC TOOL-CHEST $29.95
   BUSTER.CMD: pretty printer
   LINEXREF.BAS: line cross-referencer
   REMPAC.BAS, SPCPAC.BAS, COMPAC.BAS: remove superfluous code
   STRIP.BAS: superfluous line-numbers stripper
2. FLEX, SK*DOS UTILITIES KIT $39.99
   CATS. CMD: alphabetically-sorted directory listing
   CATD.CMD: date-sorted directory listing
   COPYSORT.CMD: file copy, alphabetically
   COPYDATE.CMD: file copy, by date-order
   FILEDATE.CMD: change file creation date
   INFO.CMD (& INFOGMX.CMD): tells disk attributes & contents
   RELINK.CMD (& RELINK82): re-orders fragmented free chain
   RESQ.CMD: undeletes (recovers) a deleted file
   SECTORS.CMD: show sector order in free chain
   XL.CMD: super text lister
3. ASSEMBLERS/DISASSEMBLERS UTILITIES $39.95
   LINEFEED.CMD: 'modularise' disassembler output
   MATH.CMD: decimal, hex, binary, octal conversions & tables
   SKIP.CMD: column stripper
4. WORD - PROCESSOR SUPPORT UTILITIES $49.95
   FULLSTOP.CMD: checks for capitalization
   BSTYCIT.BAS (.BAC): Stylo to dot-matrix printerr
   NECPRINT.CMD: Stylo to dot-matrix printer filter code
5. UTILITIES FOR INDEXING $49.95
   MENU.BAS: selects required program from list below
   INDEX.BAC: word index
   PHRASES.BAC: phrase index
   CONTENT.BAC: table of contents
   INDXSORT.BAC: fast alphabetic sort routine
   FORMATER.BAC: produces a 2-column formatted index
   APPEND.BAC: append any number of files
   CHAR.BIN: line reader

BASIC09 TOOLS consist of 21 subroutines for Basic09. 6 were written in C Language and the remainder in assembly. All the routines are compiled down to native machine code which makes them fast and compact.

1. CFILL -- fills a string with characters
2. DPEEK -- Double peek
3. DPOKE -- Double poke
4. FPOS -- Current file position
5. FSIZE -- File size
6. FTRIM -- removes leading spaces from a string
7. GETPR -- returns the current process ID
8. GETOPT -- gets 32 byte option section
9. GETUSR -- gets the user ID
10. GTIME -- gets the time
11. INSERT -- insert a string into another
12. LOWER -- converts a string into lowercase
13. READY -- Checks for available input
14. SETPRIOR -- changes a process priority
15. SETUSR -- changes the user ID
16. SETOPT -- set 32 byte option packet
17. STIME -- sets the time
18. SPACE -- adds spaces to a string
19. SWAP -- swaps any two variables
20. SYSCALL -- system call
21. UPPER -- converts a string to uppercase

For OS-9 - $44.95 - Includes Source Code
See Review in January 1987 issue of 68 Micro Journal

```
                        * RELEASE CODE - CHECK FOR
                        * FUNCTION KEY RELEASE

01AF A1 AA      relcod  cmpa   #$AA              shift key release (R)
01B1 27 14              beq    cshift
01B3 A1 B6              cmpa   #$B6              shift key release (L)
01B5 27 10              beq    cshift
01B7 A1 9D              cmpa   #$9D              control key release
01B9 27 08              beq    ccontl
01BB A1 B8              cmpa   #$B8              alt key release
01BD 26 ED              bne    readm2            return to polling
01BF 19 10              bclr   4,kstate clear    down alt key
01C1 20 E9              bra    readm2            return
01C3 15 10      ccontl  bclr   2,kstate
01C5 20 E5              bra    readm2
01C7 11 10      cshift  bclr   0,kstate
01C9 20 E1              bra    readm2            return

                        * MUST BE ORDINARY CODE - GO TO
                        * ONE OF THE FOLLOWING TABLES

01CB BE 13      sortc   ldx    data              get data fo KB
01CD 06 10 74           brset  3,kstate,numlok   go if numlock
01D0 04 10 73           brset  2,kstate,ctrlb    g if control key
01D3 00 10 72           brset  0,kstate,shiftb   use shifted table

                        * NO FUNCTION KEY IN USE - MUST
                        * BE ORDINARY CODE. CHECK IF KEY USED
                        * IS FOR CURSOR OPERATION.

01D6 B6 13              lda    data
01D8 A1 37              cmpa   #$37              is it xmit [PrtSc]
01DA 26 05              bne    ncurs1
01DC A6 04              lda    #$04
01DE CC 0147            jmp    putchr
01E1 A1 48      ncurs1  cmpa   #$48              is it ^
01E3 26 05              bne    ncurs2
01E5 A6 05              lda    #$05
01E7 CC 0147            jmp    putchr
01EA A1 49      ncurs2  cmpa   #$49              is it pg up
01EC 26 05              bne    ncurs3
01EE A6 09              lda    #$09
01F0 CC 0147            jmp    putchr
01F3 A1 4B      ncurs3  cmpa   #$4B              is it <--
01F5 26 05              bne    ncurs4
01F7 A6 0A              lda    #$0A
01F9 CC 0147            jmp    putchr
01FC A1 47      ncurs4  cmpa   #$47              is it home
01FE 26 05              bne    ncurs5
0200 A6 06              lda    #$06
0202 CC 0147            jmp    putchr
0205 A1 4D      ncurs5  cmpa   #$4D              is it -->
0207 26 05              bne    ncurs6
0209 A6 02              lda    #$02
020B CC 0147            jmp    putchr
020E A1 50      ncurs6  cmpa   #$50              is it v
0210 26 05              bne    ncurs7
0212 A6 07              lda    #$07
0214 CC 0147            jmp    putchr
0217 A1 51      ncurs7  cmpa   #$51              is it pg dn
0219 26 05              bne    ncurs8
021B A6 0B              lda    #$0B
021D CC 0147            jmp    putchr
0220 A1 52      ncurs8  cmpa   #$52              is it insert
0222 26 05              bne    ncurs9
0224 A6 01              lda    #$01
0226 CC 0147            jmp    putchr
0229 A1 53      ncurs9  cmpa   #$53              is del
022B 26 05              bne    ncursa
022D A6 03              lda    #$03
022F CC 0147            jmp    putchr
0232 A1 4F      ncursa  cmpa   #$4F              is it end [break]
0234 26 05              bne    ncursb
0236 A6 00              lda    #$00
0238 CC 0147            jmp    putchr
023B A1 4C      ncursb  cmpa   #$4C              is it form (5)
023D 26 0B              bne    ordkey
023F A6 08              lda    #$08
0241 CC 0147            jmp    putchr

0244 20 34      numlok  bra    numlk1
0246 20 17      ctrlb   bra    ctrlbb
0248 20 1C      shiftb  bra    shftb

                        * NORMAL TABLE, IF CAPS LOCKED ACTIVATED
                        * CHANGE a-z to A-Z

024A D6 028F    ordkey  lda    normal,x          ascii code normal
024D 02 10 03   mtetpp  brset  1,kstate,capslk   go set caps
0250 CC 013D    msend   jmp    putkey            not caps send key

0253 A1 61      capslk  cmpa   #$61              convert to upper
0255 25 F9              blo    msend
0257 A1 7A              cmpa   #$7A
0259 22 F5              bhi    msend
025B A4 DF              anda   #$DF              change it
025D 20 F1              bra    msend

                        * CONTROL KEY TABLE, USED IF KEY
                        * AND CONTROL KEY ARE PUSHED SIMULT.

025F BE 13      ctrlbb  ldx    data
0261 D6 034F            lda    contrl,x
0264 20 EA              bra    msend

                        * SHIFT KEY TABLE, USED IF KEY
                        * AND SHIFT KEY ARE PUSHED SIMULT.

0266 BE 13      shftb   ldx    data
```

```
0268 D6 02EF            lda   shiftt,x
026B 03 10 E2   testls  brclr 1,kstate,msend      send it if caps on
026E A1 41              cmpa  #$41
0270 25 DE              blo   msend
0272 A1 5A              cmpa  #$5A
0274 22 DA              bhi   msend
0276 AA 20              ora   #$20
0278 20 D6              bra   msend

                * NUMBER LOCK ON.

027A 04 10 0D   numlkl  brset 2,kstate,numctr
027D 00 10 05           brset 0,kstate,numshi
0280 D6 03AF            lda   numnor,x
0283 20 C8              bra   mtetpp              now go check for caps lk

                * USE TABLE NUMBER TABLE NUM SHIFT

0285 D6 040F    numshi  lda   numstb,x
0288 20 E1              bra   testls

                * USE TABLE CONTROL TABLE NUM SHIFT

028A D6 046F    numctr  lda   numrol,x
028D 20 C1              bra   msend

                *****
                * LOOK-UP TABLES
                *

                * NORMAL MODE (LOWER CASE)
028F 00         normal  FCB   0
0290 1B 31 32 33        FCC   $1B,"1234567890-=",$08
0294 34 35 36 37
0298 38 39 30 2D
029C 3D 08
029E FF 71 77 65        FCC   $FF,"qwertyuiop[]",$0D
02A2 72 74 79 75
02A6 69 6F 70 5B
02AA 5D 0D
02AC FF 61 73 64        FCC   $FF,"asdfghjkl;'",$6D
02B0 66 67 68 6A
02B4 6B 6C 3B 27
02B8 60
02B9 FF 5C 7A 78        FCC   $FF,"\zxcvbnm,./",$FF,"**"
02BD 63 76 62 6E
02C1 6D 2C 2E 2F
02C5 FF 2A
02C7 FF 20 FF 81        FCB   $FF,$20,$FF,$81,$82,$83,$84,$85
02CB 82 83 84 85
02CF 86 87 88 89        FCB   $86,$87,$88,$89,$8A,$FF,$DB,$10
02D3 8A FF DB 10
02D7 01 0E 2D 04        FCB   $01,$0E,$2D,$04,$35,$09,$2B,$03
02DB 35 09 2B 03
02DF 02 0F 19 1A        FCB   $02,$0F,$19,$1A,$FF,$FF,$FF,$FF
02E3 FF FF FF FF
02E7 FF FF FF FF        FCB   $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
02EB FF FF FF FF

                * SHIFT MODE (UPPER CASE)
02EF 00         shiftt  FCB   0
02F0 1B 21 40 23        FCC   $1B,"!@#$%^&*()_+",$08
02F4 24 25 5E 26
02F8 2A 28 29 5F
02FC 2B 08
02FE FF 51 57 45        FCC   $FF,"QWERTYUIOP{}",$0D
0302 52 54 59 55
0306 49 4F 50 7B
030A 7D 0D
030C FF 41 53 44        FCC   $FF,'ASDFGHJKL:"~'
0310 46 47 48 4A
0314 4B 4C 3A 22
0318 7E
0319 FF 7C 5A 58        FCC   $FF,$7C,"ZXCVBNM<>?",$FF,$DE
031D 43 56 42 4E
0321 4D 3C 3E 3F
0325 FF DE
0327 FF 20 FF 8B        FCB   $FF,$20,$FF,$8B,$8C,$8D,$8E,$8F
032B 8C 8D 8E 8F
032F 90 91 92 93        FCB   $90,$91,$92,$93,$94,$FF,$DC
0333 94 FF DC
0336 37 38 39 2D        FCC   "789-456+1230.",$FF,$FF,$FF,$FF
033A 34 35 36 2B
033E 31 32 33 30
0342 2E FF FF FF
0346 FF
0347 FF FF FF FF        FCB   $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
034B FF FF FF FF

                * CONTROL MODE
034F 00         contrl  FCB   0
0350 1B FF 00 FF        FCB   $1B,$FF,$00,$FF,$FF,$FF,$1E,$FF
0354 FF FF 1E FF
0358 FF FF FF 1F        FCB   $FF,$FF,$FF,$1F,$FF,$7F,$FF,$11
035C FF 7F FF 11
0360 17 05 12 14        FCB   $17,$05,$12,$14,$19,$15,$09,$0F
0364 19 15 09 0F
0368 10 1B 1D 0A        FCB   $10,$1B,$1D,$0A,$FF,$01,$13,$04
036C FF 01 13 04
0370 06 07 08 0A        FCB   $06,$07,$08,$0A,$0B,$0C,$FF,$FF
0374 0B 0C FF FF
0378 FF FF 1C 1A        FCB   $FF,$FF,$1C,$1A,$18,$03,$16,$02
037C 18 03 16 02
0380 0E 0D FF FF        FCB   $0E,$0D,$FF,$FF,$FF,$FF,$DF,$FF
0384 FF FF DF FF
0388 20 FF 95 96        FCB   $20,$FF,$95,$96,$97,$98,$99,$9A
038C 97 98 99 9A
0390 9B 9C 9D 9E        FCB   $9B,$9C,$9D,$9E,$FF,$DD,$0C,$FF
0394 FF DD 0C FF
0398 FF FF FF FF        FCB   $FF,$FF,$FF,$FF,$FF,$FF,$06,$FF
039C FF FF 06 FF
03A0 16 FF FF FF        FCB   $16,$FF,$FF,$FF,$FF,$FF,$FF,$FF
03A4 FF FF FF FF
03A8 FF FF FF FF        FCB   $FF,$FF,$FF,$FF,$FF,$FF,$FF
03AC FF FF FF
```

```
                    * NUMBER LOCK NORMAL MODE (LOWER CASE)
03AF 00             numnor  FCB     0
03B0 1B 31 32 33            FCC     $1B,"1234567890-=",$08
03B4 34 35 36 37
03B8 38 39 30 2D
03BC 3D 08
03BE FF 71 77 65            FCC     $FF,"qwertyuiop[]",$0D
03C2 72 74 79 75
03C6 69 6F 70 5B
03CA 5D 0D
03CC FF 61 73 64            FCC     $FF,"asdfghjkl;'",$60
03D0 66 67 68 6A
03D4 6B 6C 3B 27
03D8 60
03D9 FF 5C 7A 78            FCC     $FF,"\zxcvbnm,./",$FF,"*"
03DD 63 76 62 6E
03E1 6D 2C 2E 2F
03E5 FF 2A
03E7 FF 20 FF 81            FCB     $FF,$20,$FF,$81,$82,$83,$84,$85
03EB 82 83 84 85
03EF 86 87 88 89            FCB     $86,$87,$88,$89,$8A,$FF,$FC,$37
03F3 8A FF FC 37
03F7 38 39 2D 34            FCB     $38,$39,$2D,$34,$35,$36,$2B,$31
03FB 35 36 2B 31
03FF 32 33 30 2E            FCB     $32,$33,$30,$2E,$FF,$FF,$FF,$FF
0403 FF FF FF FF
0407 FF FF FF FF            FCB     $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
040B FF FF FF FF

                    * NUMBER LOCK SHIFT MODE (UPPER CASE)
040F 00             numstb  FCB     0
0410 1B 21 40 23            FCC     $1B,"!@#$%^&*()_+",$08
0414 24 25 5E 26
0418 2A 28 29 5F
041C 2B 08
041E FF 51 57 45            FCC     $FF,"QWERTYUIOP{}",$0D
0422 52 54 59 55
0426 49 4F 50 7B
042A 7D 0D
042C FF 41 53 44            FCC     $FF,'ASDFGHJKL:"~'
0430 46 47 48 4A
0434 4B 4C 3A 22
0438 7E
0439 FF 7C 5A 58            FCC     $FF,$7C,"ZXCVBNM<>?",$FF,$FE
043D 43 56 42 4E
0441 4D 3C 3E 3F
0445 FF FE
0447 FF 20 FF 8B            FCB     $FF,$20,$FF,$8B,$8C,$8D,$8E,$8F
044B 8C 8D 8E 8F
044F 90 91 92 93            FCB     $90,$91,$92,$93,$94,$FF,$FB
0453 94 FF FB
0456 10 01 0E 2D            FCB     $10,$01,$0E,$2D,$04,$35,$09,$2B
045A 04 35 09 2B
045E 03 02 0F 19            FCB     $03,$02,$0F,$19,$1A,$FF,$FF,$FF
0462 1A FF FF FF
0466 FF FF FF FF            FCB     $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
046A FF FF FF FF
046E FF
```

```
                    * NUMBER LOCK CONTROL MODE
046F 00             numrol  FCB     0
0470 1B FF 00 FF            FCB     $1B,$FF,$00,$FF,$FF,$FF,$1E,$FF
0474 FF FF 1E FF
0478 FF FF FF 1F            FCB     $FF,$FF,$FF,$1F,$FF,$7F,$FF,$11
047C FF 7F FF 11
0480 17 05 12 14            FCB     $17,$05,$12,$14,$19,$15,$09,$0F
0484 19 15 09 0F
0488 10 1B 1D 0A            FCB     $10,$1B,$1D,$0A,$FF,$01,$13,$04
048C FF 01 13 04
0490 06 07 08 0A            FCB     $06,$07,$08,$0A,$0B,$0C,$FF,$FF
0494 0B 0C FF FF
0498 FF FF 1C 1A            FCB     $FF,$FF,$1C,$1A,$18,$03,$16,$02
049C 18 03 16 02
04A0 0E 0D FF FF            FCB     $0E,$0D,$FF,$FF,$FF,$FF,$FD,$FF
04A4 FF FF FD FF
04A8 20 FF 95 96            FCB     $20,$FF,$95,$96,$97,$98,$99,$9A
04AC 97 98 99 9A
04B0 9B 9C 9D 9E            FCB     $9B,$9C,$9D,$9E,$FF,$FA,$0C,$FF
04B4 FF FA 0C FF
04B8 FF FF FF FF            FCB     $FF,$FF,$FF,$FF,$FF,$FF,$06,$FF
04BC FF FF 06 FF
04C0 16 FF FF FF            FCB     $16,$FF,$FF,$FF,$FF,$FF,$FF,$FF
04C4 FF FF FF FF
04C8 FF FF FF FF            FCB     $FF,$FF,$FF,$FF,$FF,$FF,$FF
04CC FF FF FF
                    *
                    *
                    * MASK OPTION REG
                    * SET FOR XTAL

0784                        org     $0784
0784 07             moa     fcb     %00000111

                    * INTERUPT VECTOR TABLE
                    *
07F8                        ORG     $07F8
07F8 0130           TIMERV  FDB     time
07FA 0121           EXTINT  FDB     go
07FC 0100           SYSSWI  FDB     reset
07FE 0100           VRESET  FDB     reset
                            END
```

0 ERROR(S) DETECTED

SYMBOL TABLE:

```
ENTINT 07FA  SYSSWI 07FC  TIMERV 07F8  VRESET 07FE  capslk 0 53  ccount 01C3  conrci 034F  count  0012  cshift 01C7
ctrlb  0246  ctrlbb 025F  data   0013  ddra   0004  ddrb   0005  ddrc   0006  go     0121  kstate 0010  meltss 019F
mnaplo 019B  mnapol 0194  mctrls 018C  mnamlo 01AA  mnumml 01A3  moa    0784  novel  0163  nmend  0250  nshift 0190
ntstpp 024D  nours1 01E1  nours2 01EA  nours3 01F3  nours4 01FC  nours5 0205  nours6 020E  nours7 0217  nours8 0220
nours9 0229  noursa 0232  noursb 023B  normal 028F  numctr 028A  numlk1 027A  numlok 0244  numnor 03AF  numrol 046F
numslt 0285  numstb 040F  orrkey 024A  out    0161  porta  0000  portb  0001  portc  0002  putchr 0147  putkey 013D
readl  0155  readml 0151  rmode2 01AC  relcod 01AF  reset  0100  shftb  0 66  shftcb 0240  shiftt 02EF  sorte  01CB
store  0011  tcr    0009  tdr    0008  tmscls 026B  time   0130
```

***

A REVIEW OF WETPAINT
New Clip Art for the Mac

By: James E. Law
1806 Rock Bluff
Hixson, TN 37343

Mac-Watch

With the current information explosion, there is a proliferation of paper assaulting each of us. With newsletters, junk mail, advertisements, and such like competing for attention, how can one ensure his or her work gets the attention it deserves? Perhaps part of the answer is in the old adage, "a picture is worth a thousand words." To this end, a number of companies have produced clip art for the Macintosh users.

The latest such offering is WetPaint by Dobi-Check Software, Inc. WetPaint contains over 2.2 megabytes of art on 6 double-sided disks. It also contains two desk assessories (Art Roundup and Pattern Mover), and one font (San Quentin).

This review initially considered Rev. 1 to Art Roundup. The large number of bugs and system crashes with the version resulted in a call to Dobi-Check who said that problems with Rev. 1 had resulted in Rev. 1.1. About two weeks after disk 1A was returned, an updated disk was received.

Art for Arts Sake

Wet Art is recorded in MacPaint (i.e., PNTG) format and can be opened by MacPaint, Super Paint, Full Paint, or other programs that work with this format. Wet Art clip art appears on 87 - 8 by 10 inch pages. Each page contains from 1 to more than 70 images. Overall, this collection includes well over 1,000 separate images. The great variety presented ensures that there will be something useful to almost everyone. Some of the material covered includes:

- Trains, cars, boats, trucks, and bicycles
- Animals of every description
- Seasonal pictures to spice up your holiday cards and party announcements
- Borders
- Western art: guns, cowboys, saddles, rodeo scenes
- Maps
- Business art: computers, Mastercard, office equipment
- Food
- Clocks and watches
- Minature icons: flags, international symbols, semaphores
- Men and women, and
- Much more

According to Dubi-Check, this art, with only minor exceptions, is new and not previously published.

A picture of the author or a sample of WetPaint clip art?

A number of the pictures are designed to be personalized by the user. For example, tee shirts are set up to put the users unique message on them, a meeting notice in the form of a big clock can be modified to show the starting time of the users meeting, and a special large font is provided to prepare customized license plates.

Many pages contain helpful suggestions on how to modify and use the various images.

Overall the quantity, variety, and quality of clip art provided makes for a solid value.

Art Roundup

Art roundup is a desk assessory which allows the Mac user to open clip art, MacPaint, or Full Paint files; select and modify images; and transfer those images to the clipboard without leaving the application in use. This means that while preparing a document in MacWrite, Art Roundup can be used to quickly review art files, select an image, and transfer that image through the clip board into the McWrite document. This program performs essentially the same function as Art Grabber or Quick Paint.

Upon selecting Art Roundup from the desk assessory menu, a new menu appears with a number of tools and options. Selecting "Open Paint File" displays all art files on the current disk which can be read by Art Roundup. Selecting a

specific art file results in the related images being displayed very quickly (less than 2 seconds in my system). A "Show Page" option allows the entire 8 by 10 inch page to be viewed. A rectangular can be moved to select the area of the page to be viewed. (This is identical to the show page feature in MacPaint.)

The scroll bars can be used to further adjust the position of an image in the viewing area, but this works extremely slow. A far faster solution is to select and use the grabber hand.

An eraser is provided to modify images or to clean up around the image to be selected. The image can then be selected using the selection marquee or lasso. Images selected with the marquee can be inverted, flipped horizontal, or flipped vertical.

Art Roundup supports the use of the MacPlus cursor keys for opening and selecting art files. Several keyboard shortcuts are also provided to speed things up for users.

Art Roundup is a very handy tool and is a great companion to WetPaint's clip art. It does what it is supposed to do and does it fast. Art Roundup is temperamental, however, and even Rev. 1.1 hung up fairly often, particularly if an effort was made to select and copy a large image. While Art Roundup occupies less than 10K of RAM, another 50K is occupied when a paint document is opened. Perhaps some problems encountered by the reviewer were due to tight memory situations. These problems may be avoided by closing a window and trying again or by selecting the art in sections to be pieced together in the receiving application.

Pattern Mover

Pattern Mover is a desk accessory which allows the editing of patterns and the transfer of single files or entire pallets between different Full Paint or MacPaint files. It will be especially helpful to Mac artists who create their own patterns for use in Macpaint, Superpaint, or Full paint and want to move these patterns between documents or between applications. Pattern Mover was not reviewed.

System Requirements

WetPaint clip art can be used on any Macintosh including the 128K version. At least 512K will be necessary, however, to use Art Rounup.

Conclusion

The combination of Wet Paint's high quality art and the art accessing and handling capabilities of Art Roundup provides a solid value. In spite of the quirks in Art Roundup, this package is highly recommended.

MACINTOSH TRENDS... Major New Macintosh Software Offerings From Apple

Apple has announced two major new software offerings for the Macintosh. The first, called **HyperCard**, is already on dealers' shelves.This program is a lot harder to explain than it is to use. HyperCard consists of a number of "stacks" each of which is composed of a number of related "cards. " For example, there may be a stack of "to-do" lists with a separate card for each day; a stack of inventory cards with a separate card for each item in an art collection, or a stack of sheets from an instruction manual will each card being a separate page. The cards may be related to other cards, even those in other stacks in any manner desired. Each card is up to the size of the Macintosh screen. Lest you think that HyperCard is simply a new relational data base program, read on.

The cards in HyperCard may be equipped with powerful buttons which may be used to branch off to related information, dial a telephone, activate a subroutine involving sound, animation, and/or graphics, or launch another application. It is also equipped with extensive graphics capabilities for preparing professional- looking cards.

This software accompanied with a very clear and thorough manual which is the same size as the one that comes with a new Macintosh.

HyperCard is shipped on 4 disks which contain numerous examples of how it may be used. Those examples are intended to be the jumping off point as users create their own HyperCard application. HyperCard requires a 1MB of RAM to run. It will be provided free with Mac sold in the future but is available for existing Mac owners for $49.00.

HyperCard is so powerful that its applications will be limited more by the creativity of users than by the capabilities of the software. In upcoming months I believe you will see many HyperCard applications being offered by users groups and software companies.

The other new Apple offering is **MultiFinder**. This software , which was previously called Juggler, was scheduled to be shipped to dealers in September. MultiFinder is the first step toward a multi-tasking environment for the Macintosh. It allows windows from different applications to be open at once. While all windows are visible, only one window is active at a time. When a given window is selected, the associated application is instantaneously acttvated.

As initially offered, MultiFinder does not offer true multi-tasking capabilities. It will allow future programs to be designed to cause some activities such as printing or downloading files to procced in the background while an application runs.

Like HyperCard, MultiFinder will be free with future Macs and will be available to existing owners at a cost of $49.00.

Pascal A Tutorial

By: Robert D. Reimiller
Certified Software Corp
616 Camino Caballo
Nipomo, CA 93444
805 929-1359

In any language you need some way to get down to the operating system call level, assuming your program runs on an operating system. One of the most common methods is the use of external procedure or function calls.

The operating systems I have experience with generally fall into three categories as far as system calls are concerned.

1) Parameters passed in registers almost exclusively, error status returned in a register, examples are OS-9 and PDOS.
2) A pointer register is used to pass parameters, it may point to a parameter block, error status returned in a register, examples are CP/M-68K and VERSAdos.
3) All parameters passed on the stack, error status returned in a register, an example is GEMDOS on the ATARI ST.

Note that only option 3 is easily handled by a compiler, since all parameters are passed on the stack anyway, no major changes are required, and there is no worry about random use of registers.

Since we are talking about OS-9 here, we will only cover option 1. These system calls generally have the task of moving parameters off of the stack into the appropriate registers, doing the system call, and then setting any return values back on the stack so the high level language can access them.

As an example of mixing standard Pascal I/O with OS-9 system calls, lets look at a program that has the job of restoring the module checksum and CRC. The program is designed to allow multiple modules per file (such as a boot file) and will automatically make the distinction between OS-9/6809 and OS-9/68000 module formats.

The program :

```
Program validate {$v1,1 os9 call example} ;
  type
    {pascal device descriptor for file of byte}
    descr = record
              p_mode : byte ;
              p_err : byte ;
              p_drv : longhex ;
              p_elnt : integer ;
              p_elmx : byte ;
              p_path : hex
            end ;
  var
    f : file of byte ;
    m68k : boolean ; {true if 68000 format}
    idx, parity9 : byte ;
    parityk, err : hex ;
    scount, count, modsize : longhex ;
    p : ^descr ;
    crc : longhex ; {crc accumulator}
    header9 : array [$0 .. $8] of byte ; {6809
                                  header buffer}
    headerk : array [$0 .. $$17] of hex ;
                             {68000 header buffer}
    buf : array [$0..$3FF] of byte ; {1k data
                                      buffer}

{$I/dd/pdef/os9calls access external procedure
                                    definitions}

  begin
    {$I+}
    open (f, cline(1), update) ; {use 1st
                        command line parameter}
    p := addr(f) ; {get address of variable f}
    while not eof (f) do
      begin {scan for header}
        read (f, header9[0]) ;
        if (header9[0] = $87) or
           (header9[0] = $4a)
          then
            begin {have valid first byte, look
                                    for second}
              read (f, header9[1]) ;
              if (header9[1] = $CD) and
                  (header9[0] = $87)
                then
                  m68k := false {6809 header
                                      format}
                else
                  if (header9[1] = $FC) and
                                      (header9[0] = $4a)
                    then
                      m68k := true {68000
                                 header format}
```

```
        else
          exit ; {not a loadable
                          module}
    crc := $ffffff ; {initialization
                       value}
    if m68k
      then
        begin {use words}
          headerk[0] :=
              hex(header9[0]) << 8 +
              hex(header9[1]) ;
          for idx := 1 to 22 do
            begin
              read (f, parity9) ;
              {build header words}
              headerk[idx] :=
                 hex(parity9) << 8 ;
              read (f, parity9) ;
              headerk[idx] :=
                 headerk[idx] +
                 hex(parity9) ;
            end ;
          parityk := 0 ;
          for idx := 0 to 22 do
            {calculate parity}
            parityk := parityk eor
                       headerk[idx] ;
          headerk[23] := not parityk;
          {set parity}
          write (f,
            byte(headerk[23] >> 8)) ;
          write (f,
             byte(headerk[23])) ;
          {and update file}
          {calculate module size
              (remainder)}
          modsize :=
            longhex(headerk[2]) << 16
              + longhex(headerk[3])
              - $33 ;
          {do crc over parity area of
                          module}
          err := f_crc (crc,
                 addr(headerk), $30)
        end
      else
        begin
          {similar for 6809 modules,
           except we work on
           bytes instead of words}
          for idx := 2 to 7 do
            read (f, header9[idx]) ;
          parity9 := 0 ;
          for idx := 0 to 7 do
            parity9 := parity9 eor
                  header9[idx] ;
          header9[8] := not parity9 ;
          write (f, header9[8]) ;
          modsize :=
              longhex(hex(header9[2])
              << 8 +
              hex(header9[3]) - $C) ;
          err := f_crc (crc,
                 addr(header9), 9) ;
        end ;
    {calculate crc using 1K chunks}
    for count := 1 to modsize>>10 do
      begin
        scount := 1024 ;
        err := i_read (addr(buf),
                 scount, p^.p_path) ;
        err := f_crc (crc, addr(buf),
                 scount)
      end ;
    {calculate crc over remainder of
                          file}
        scount := modsize and 1023 ;
        if scount <> 0
          then
            begin
              err := i_read (addr(buf),
                     scount, p^.p_path) ;
              err := f_crc (crc,
                     addr(buf), scount)
            end ;
        {update crc for this module}
        write (f, not byte(crc >> 16)) ;
        write (f, not byte(crc >> 8)) ;
        write (f, not byte(crc))
      end
  end ;
close (f)
  end .
```

Pascal extension notes :

1) Extra data types : byte = same as char, one byte unsigned, hex = 2 byte unsigned, longhex = 4 byte unsigned.

2) a $ preceding a value indicates that the value is taken as hexadecimal.

3) a # preceding a value indicates that the value is taken as a byte value.

4) eor is the exclusive or operation

5) >> is shift right (shift count is on the right of the operand), likewise, << is shift left.

6) the convention of "<data_type> (<value>)" is used to convert the value to the data type.

7) the exit statement exits the enclosing loop (in this case the while loop).

8) the addr function returns the address of a function or procedure.

9) the open procedure has it's second parameter the file name, as it's third parameter, the data direction (input, output, or update).

10) Operators such as not, and, or, eor are logical if the operand is boolean, bitwise for all others.

In the file os9calls, f_crc has the following definition :

{On input acc is the crc accumulator, count is the data byte count, and ptr is the ADDRESS of the data block. On output acc will be the updated crc accumulator}

function f_crc (var acc : longhex ; ptr, count : longhex) : hex ; external ;

i_read is defined as follows :

{On input buf is the ADDRESS of the data buffer, count is how many bytes you want to read, and path is the path number. On output count will be the actual number of bytes read.}

function i_read (buf : longhex ; var count : longhex ; path : hex) :
hex ; external ;

In the os9calls file are about 45 other definitions for OS-9 system calls. Implementing the more obscure system calls is left as an exercise for the reader!

OS-9 does all file accesses using a 2 byte "path" number. When the file is opened using the pascal "open" procedure, the path number returned by the operating system is stored in the file variable in the field "p_path". We use this path number when we do the i_read call. The i_read call is used since it is capable of reading any arbitrary number of bytes of data, rather than using the file variable definition of 1 byte per record.

The example program is entered using the editor, and then the compiler is called to compile from the edit buffer to check for syntax. Also noted is that the stack size is $46E bytes.

The linkage creator is called to setup the control files for this little project :

```
    $ lc validate
    OmegaSoft Linkage Creator Version 1.21
    Copyright 1987 by Certified Software
Corporation
    Global stack size not determined
    auto setup ? y
    Process 2 command line option ? n
    Stack, heap, and Varib size : 600
    Library directory name : /dd/
    Other pascal files :
    Other assembly files : /dd/pdef/os9code
    Other assembly files :
    Other library files :
    Linker command line info : validate.mp
    Map options : f
    Include files : /dd/pdef/os9calls
    Include files :
    Debugger options :
    Target debugger options :
    Compiler options : -d
    Compiler options : -r
    Compiler options : -lp62ftw96 /p
    Compiler options :
    Assembly options : -o
    Assembly options :
    Editor options :
    OmegaSoft 68000 Assembler version 1.21
    Copyright 1987 by Certified Software
Corporation
    Errors : 0  Code : 00AE  Data : 0000
Varib : 0000  Table 14 of 944
    $ ps validate
```

Now in the pascal shell, we can compile it using the -D option, and then run the debugger. When we are in the pascal mode of the debugger we need to set in the file name we want the program to use. This is because the programs uses "cline(1)" as the file name. "Cline" is a function that returns a command line parameter, 0 returns the entire command line, 1 returns the first parameter, 2 returns the second, etc.

```
<P> sc
<$> testfile
<P> g
```

Assuming we have a file called testfile (I just copied a sample file from the command directory) it would reset the module(s) checksum and CRC to the correct values in that file and exit.

This particular program is probably of not much use, since the only way the checksum and crc should be modified is if the file is patched, and the OmegaSoft patch utility includes a command to restore the checksum and crc. In fact, this example was derived from the applicable procedure in the patch command (why write something new when you can borrow code from something you did before?).

Next time we will look at doing multi-tasking under OS-9/68000.

EOF

FORTH

A Tutorial Series

By: R. D. Lurie
9 Linda Street
Leominister, MA 01543

INSTALLING FIG-FORTH

Some of you may be considering joining in the pleasures of FORTH programming by installing FIG-FORTH, so I thought that it would be a good idea to report on my experiences with the 6809 version.

FORTH is a lot of fun and easy to use, if you start off right. But FORTH can be a killer if you get off on the wrong foot! There are two ways to get FIG-FORTH for the 6809, either you can buy the printed listing from FIG for $15 and the installation manual for another $15, or you can purchase one of the reasonably priced systems from several vendors in a ready-to-use form. I recommend that you save yourself a lot of agrivation by buying a system, unless you know how to use the FLEX assembler and have a good editor.

The version of FIG-FORTH you get from FIG has no instructions on how to install it in any machine. They assume that you already know how to get a complicated program up and running without any help from anywhere. The "installation manual" was written for a 6502 processor, and is really of no help, except for the primitive FORTH editor it contains and the FIG glossary (which includes words not found in FIG-FORTH for the 6809, and vice versa).

FIG-FORTH is the public domain version of the FORTH sold for years by TALBOT MICROSYSTEMS (who did advertise in 68' MJ, but that is not why I am writing this!). I would guess that their manual would be the ideal choice to go with 6809 FIG-FORTH. In any case, you will need at least one good book on FORTH before you can do much with FIG-FORTH.

In order to use FIG-FORTH for the 6809 in the form supplied by FIG, you must have the items listed in figure 1.

```
1. A 6809 computer with the FLEX (or SK*DOS?) operating
   system.
2. At least one disk drive.
3. A terminal communicating through an ACIA.
4. A minimum of 16k of RAM, beyond that required for DOS.
5. An editor for entering the program to disk.
6. A macro assembler (the FLEX assembler is assumed).
```

Figure 1. The minimum hardware/software requirements for installing FIG-FORTH.

As you can see from item #3, a CoCo cannot be used, even if one of the versions of FLEX or SK*DOS has been installed. Unless you are prepared to do a lot of assembly language work, don't kid yourself into trying to force-feed FIG-FORTH into the CoCo.

Even if you have all of the items on the list, be prepared to spend several (!!) hours typeing and debugging, before the listing will even assemble. I found one error in the listing; there is a typographical error in the macro for TICK . This can be found at the top of page 38. The correct form is

```
1-|WORDM 1,"",IMMEDIATE
```

Somehow, the two quotation marks got left out when the listing was printed. I am embarassed everytime I think about how long it took me to find that this was what was fouling up the assembler.

After reading all of this, you are now probably convinced that you do not want to try to install FIG-FORTH. I hope that I have not laid on the cautions too thickly, since FIG-FORTH does work, and it works very well! You just have to be aware of the potential pitfalls before you start.

WHY CONSIDER FIG-FORTH?

You may wonder why I even bring up the subject of FIG-FORTH, if it is so hardware limited and such a pain to get going. My reason is that it is the only source I know of for a complete 6809 FORTH source-code listing which does not require a meta-compiler or other specialized software. If you want to customize FORTH to an unusual hardware configuration or to a minimal size for a particular situation, you must have the source-code in a form that is easy to modify and work with, and that is what you get with FIG-FORTH.

Frankly, FF9 from Wilson Federici is a much better deal for the average hobbyist, since it comes ready to run and is a greatly extended version of FORTH-83. You can easily and readily extend FF9 further, but you would have a lot of difficulty making it any smaller, since a meta-compiler was used to generate the original source-code. For most of us, that is no problem, provided you can use the "floored math" of FORTH-83.

WHY NOT USE FORTH-83?

Despite some of my rather loud rantings and ravings about the surprises in FORTH-83 for those weaned on FIG-FORTH, I do think that it really is a friendlier programming haven than FIG-FORTH. However, FORTH-83 does use floored math. "Floored" math means that any operation in integer math which has a remainder results in value truncation (not rounding) toward negative-infinity. Of course, any math scheme is floored math, but the expression is not used for the much more common convention of using zero as the floor.

MATH PROBLEMS

I am sure that there were some good arguments extolling the advantages of minus-infinity as the floor when the subject was presented to the standards committee; otherwise, it would never have been put into the FORTH-83 standard. Unfortunately, I have never knowingly encountered any of these advantages. The big disadvantage, from my point of view, is that FORTH-83 can give the WRONG ANSWER to any division problem which produces a negatively signed quotient. Presently, the only fix I know of for

this bug is to do the division as unsigned, and then force the quotient to be negative after all of the division is finished. I don't like that solution, but I don't know of any other which will work in FORTH-83. If you don't like that idea, then you must either use FIG-FORTH or FORTH-79 or, else, write you own math routines, which will not be a part of the FORTH-83 standard.

You may wonder at why this is a problem, so I will give you a simple example from FIG-FORTH and FORTH-83. The examples in

```
FIG-FORTH:
     10000  355  113  */  ==>   31415
    -10000  355  113  */  ==>  -31415

FORTH-83:
     10000  355  113  */  ==>   31415
    -10000  355  113  */  ==>  -31416
```

Figure 2. An illustration of the difference between division in FIG-FORTH and FORTH-83.

Figure 2 show that you get the same answer when you use the */ operation with positive or negative numbers with FIG-FORTH, but you get a different answer when you use the same operation in FORTH-83. I grant that the difference is small, but there should be NO difference! Consider what kind of problems you could get into when trying to move an accounting program from FIG-FORTH to FORTH-83. Sometimes I think that the people who write standards don't live in the real world (I'll grant equal time to the opposition)! Now you can see why there is a market for FIG-FORTH and FORTH-79.

FORTH IN ROM

Every once in awhile, there is a flurry of interest in putting FORTH into ROM. This is a very good idea for use in dedicated controllers, and other such devices, but it poses a problem for the general purpose computer. When FORTH is put into ROM, it is then very difficult to change it, which is the point of using ROM in the first place. However, the general purpose use of FORTH often requires extension of the dictionary with words which are unique to a particular job. Therefore, these words must either go into RAM or go into a new, special purpose ROM. Unless you want to keep a library of ROMs, it appears to me that one is better off keeping a library of disks.

Except in very special cases, FORTH requires some RAM for the Data and Return Stacks. Therefore, under normal circumstances, there is no practical way to escape using at least a few words of RAM. In fact, I cannot think of any way that a true FORTH for the 6800 or 6809 could be written which did not have a Return Stack, even if you could avoid a Data Stack by judicious use of the registers. I am sure that even the relative multitude of registers in the 68000 would not remove the necessity for some RAM, because you would still need the Return Stack. As I think about it, it occurs to me that the way 16/32 bit machines "waste" RAM, the 68000 would probably need more RAM than a 6809 to do the same job.

Some people think that putting FORTH in ROM would somehow make it run faster than the same code in RAM. For the same CPU clock speed, assuming no wait states, programs in RAM and ROM should execute at the same speed, so that there is no way for one to run faster than the other.

The only way to gain speed would be in the original loading of the FORTH. If the FORTH were in ROM instead of on a disk, you would not have to spend the loading time for the original setup each time that you turned on the computer, but there would be no gain beyond that. Program execution would stay the same.

Of course, the situation would change for a cassette tape system. Programs just do not load as fast from tape as they do from disk, so you could get pretty tired of waiting for your FORTH to load. This must be one of the reasons for BASIC in ROM on the CoCo, etc. I certainly remember the agonizing wait for SWTP 8k BASIC to load into my 6800. Under those conditions, I would even have settled for COBOL programming, if it were in ROM and I had enough RAM to go with it!?!

SINGLES TO DOUBLES

This has nothing to do with sex! Many FORTH words expect to find a double precision (32-bit) integer on the Data Stack, even though we usually do as much as possible with single precision (16-bit) integers in order to speed up program execution. The question often arises as to the best way to convert a single into a double.

I have the bad habit of simply stuffing a 0 onto the Data Stack in order to make the conversion as quickly and as painlessly as possible. This is acceptable for unsigned or positive 16-bit integers, but it is wrong for negative 16-bit integers. If the single precision number on the Data Stack is negative, then -1 is the correct value to stuff onto the stack.

The proper way to do this job is with S->D . An effective high level definition is:

```
: S->D ( n -- d ) DUP 0< IF -1 ELSE 0 THEN ;
```

You already have this word in FIG-FORTH, but it was left out of FF9. Of course, it would be better to define this word in assembly language if you need a speedy conversion. Look at Figure 3 for an assembly language definition suitable for FF9.

```
CODE  S->D     ( n -- d )
      0  #     LDD
      0  ,U    TST
      MI     IF,     COMA  COMB     THEN,
      ,--U     STD
      NEXT,
END-CODE
```

Figure 3. FF9 code definition for S->D

In case you are wondering, the definition of Figure 3 came from the FIG-FORTH source-code, and the resulting machine codes are identical. The high level definition takes 202 microseconds and the low level definition takes 41 microseconds to extend the size/sign of a negative integer. The high level definition takes 182 microseconds and the low level definition takes 37 microseconds to process a positive integer. This factor of only 4.9 times as long shows how remarkably fast a FORTH definition can be.

EXECUTION TIMES

I am in the process of running an exhaustive study on the execution speed of most, if not all, of the common FORTH functions, and I will report on them as soon as I complete the experiment. At present, I plan to compare FIG-FORTH and FF9 for the 6809 and LMI Z-80 FORTH for CP/M. If I can make it work, I will also download the Z-80 FIG-FORTH from the CP/M-SIG on CompuServe to be included with the others. I should mention that I am using a GMX 6809 cpu card operating at 1 Mhz. and a Z-80 operating at 4 Mhz in a CP/M system.

As you can imagine, this study takes a lot of time, and I have just barely started with the FF9. I am not sure how long the study will take take, since I want to compare a reasonable selection of definitions in both high level FORTH and assembly language. Furthermore, if I can manage it, I will try to throw in as many comparable operations as I can in C, BASIC, and possibly PASCAL. Whether or not this can be done depends on both my programming ability and language availability. I would be happy to hear any suggestions any of you would care to make; and constructive criticism of the testing method will be welcomed. I have no ax to grind, here, so I don't care how long it takes to finish the study, if ever, and I have no predisposition as to the results. I'll just report them as I measure them.

I mention this partly because of the surprising (to me) result I got from timing the IF ... ELSE ... THEN branching conditional in FF9. I should not have been surprised that a TRUE branch takes significantly longer than a FALSE branch (a 4:3 ratio). The reason is that a TRUE branch simply must execute more machine code than a FALSE branch, so it must take longer. This is not unique to FF9, but the ratio of times to execute the two branches must depend on the code written and the cpu used for execution.

It turns out that there is no significant difference in FF9 for the execution time of the simpler IF ... THEN conditional, whether the Boolean flag is either TRUE or FALSE. This is what I expected, intuitively, but I was amazed to discover that this is not true for the LMI Z-80 FORTH. I suppose that the Z-80 JZ/JMP command combination was used to write the definition, and this could account for the timing difference.

All of the times are based on "empty" functions, in that I have found that the times I get are additive, so that I can extract known time fractions from an experiment so that the difference can be attributed to a single operation.

CASE STATEMENT EFFICIENCY

As you may have noticed, I am partial to the CASE ... ENDCASE structure proposed by Eaker in FORTH DIMENSIONS, II/3 . I devoted considerable space to it in the December, 1986, issue of 68 'MJ, so you can refer to either place for more details on the use of this version of CASE ... ENDCASE .

I was particularly interested in seeing the results of a timing study on this subject since I do use it so much. The results are summarized in Figure 4. This definition was called from a nested

DO ... LOOP structure which executed a total of 100,000 times. The reported values are the averages extracted from each run, after the time used by the loop, itself, and loading the CONSTANT had been factored out. Figure 5 shows this structure.

```
: CASE-STUDY  ( n -- )
        CASE
     0  OF  ENDOF          \ 235 microseconds
     1  OF  ENDOF          \ 374 microseconds
     2  OF  ENDOF          \ 513 microseconds
     3  OF  ENDOF          \ 654 microseconds
     4  OF  ENDOF          \ 795 microseconds
     ( default )           \ 774 microseconds
        END-CASE ;
```

Figure 4. The "empty" CASE ... END-CASE timing experiment.

```
 3  CONSTANT  3
 4  CONSTANT  4
54  CONSTANT  54

: TEST
     TIMER-ON
     2  0  DO
       50000  0  DO
         0  CASE-STUDY      \ 0 - 4 and 54 were used here
       LOOP
     LOOP
     TIMER-OFF ;
```

Figure 5. The nested DO ... LOOP structure used to call CASE-STUDY .

In order to simplify the test, the value of the CONSTANT was changed and the definition was reloaded for each value tested.

When I started to study the test results, I found the same kind of additive time factors as I had found with other operations. After examining these results, I have calculated (and measured) that there is a minimum overhead of 214 microseconds if there is only one selector, and it is not chosen. This compares to 34 microseconds for a single IF ... ELSE ... THEN conditional taking the FALSE path. Therefore, don't use CASE ... ENDCASE for a single choice-pair, if you have any need for execution speed.

I have not had time to determine the cause for this very high overhead for CASE ... END-CASE; it will probably turn out to be something obvious, once I get a chance to study the situation. However, it is a good example for the adage about taking nothing for granted.

The CASE ... END-CASE construct is essentially only a more convenient way to program a series of IF ... ELSE ... THEN choices. It is not a jump-table! A cursory glance at the data in Figure 4 shows that you should not use the CASE ... END-CASE for any operations which must be run in the shortest possible time. Furthermore, put the most common choice at the head of the list and the least likely choice at the bottom of the list. For practical purposes, the last choice on the list and the default will be selected in about the same amount of time. Now I know why some of my programs have been running slower than I expected!

The ideal use of the CASE ... END-CASE construct is in an operation like keyboard input sorting, since there is no possible .way that you can type fast enough to outrun the computer. On the other hand, a different selection scheme should be used for time- sensitive operations, such as animated graphics. In other words, don't let programming habits cause you to make poor programming choices.

EOF

Build the GT-4 Graphic Terminal
(A Construction Project)

By:
Joseph D. Condon
8072 172nd. Street W.
Lakeville, MN 55044
Phone: 612-431-7624

Introduction

With the increased popularity of OS9 and other multiuser operating systems, there has developed a strong demand for low cost high quality ASCII terminals. Unfortunately, a good quality general purpose terminal will sometimes cost as much as the system you connect it to. Many of us own or use systems capable of supporting multiple users with only one terminal attached to them. Having a second terminal in the den or upstairs out of the basement is a luxury few of us can afford. For those of you who do not yet own a system and are considering purchasing one of those low cost single board computers, dont forget that you will have to connect a terminal to it before you can use it. The additional expense of a good quality terminal can sometimes raise the price of a single board system beyond the budget of a typical hobbyist.

If you are willing to invest some time and if the thought of wire wrapping or the smell of solder doesnt make you violently ill, I can show you how you can build your own ASCII terminal with all the bells and whistles including medium resolution graphics for a fraction of the cost of a commercially available unit.

How much of a fraction? Well that depends. You will have to purchase a monochrome video monitor with a composite video input and an IBM XT compatible keyboard. These two items will account for the majority of your expenses. With a little bargain hunting you should be able to purchase these two items for under $150.00. The remaining cost will be for components and a case to put them in. The total cost for the project should be under $250.00 if you buy everything new. If you have a well stocked junk box you may be able to reduce the cost to under $200.00.

Before committing your time and hard earned cash to this project, you probably would like to know a little more about what you will receive for your efforts. The terminal, which I will refer to as the "GT" (graphic terminal) is an asynchronous, RS232, ASCII type terminal. The terminals electronics including the power supply is designed to fit neatly into a 10"L x 7.5"W x 3.1"H case available from Jameco Electronics (part no. H2507).

The GT is capable of operating at speeds up to 19200 baud. The communication default parameters such as baud rate, parity, character data length and stop bits are switch programmable. These parameters can be changed or overridden at any time by use of a built in configuration menu which can be invoked from the keyboard. The GT's screen format is 80 characters wide by 24 lines high. The GT contains both upper and lower case character sets and supports character underlining and character reverse video. Editing functions include character insert and delete, line insert and delete, clear to end of line and clear to end of page. The GT also supports all of the standard control codes such as clear screen, home cursor, line feed, carriage return, etc. Absolute cursor positioning and invisible cursor mode are also supported.

In addition to the previous items, the GT provides medium resolution graphic capabilities. The display screen of the GT can be considered as a plane of pixels 640 wide by 240 high. Each individual pixel can be turned on, off, complimented or tested using simple ASCII escape sequences. There is also a fast line drawing function based on Bresenham's line drawing algorithm. Graphics and text can be mixed and will appear on the screen simultaneously. The GT is also capable of dumping to or loading from the host computer partial or complete screen images in the form of ASCII text records. In combination with a good quality keyboard and video monitor, the GT will make an attractive first or second terminal with excellent display quality for a minimum cost.

Because of the amount of information that needs to be covered, this construction project will be presented in two parts. The first part concentrating on the hardware and the second part concentrating on the software. The software for the GT is complete and fully functional. It is not necessary for the builder to be a programmer in order to construct a working terminal. The software source will be printed in part two of this article and will also be made available by "68 Micro Journal" in the form of a reader service disk. You the builder must have access to a "FLEX" system capable of assembling the 6809 source code and burning the object file into a 2764 type EPROM. For those of you who **absolutely** do not have access to a "FLEX" system or EPROM burner, I can supply a limited number of programmed eproms for a price of $20.00 each. I really do not have the time or facilities to burn large quantities of EPROM's but I will do it for those individuals who **absolutely** have no other means to acquire the EPROM.

The Hardware

The GT is designed around the MC68B09E micro processor. The "E" style processor is similar to the standard 6809 except that it requires externally generated E and Q clock signals. This allows for complete synchronization of the processor in relation to external events. This synchronization is required in the GT's design.

The 6809 series processor only requires access to its memory and I/O devices when the E clock signal is high or active. When the E signal is low, the processors address and data bus can be used by other devices provided the processor is isolated from the address and data bus. This type of operation is known as memory access interleaving and is especially well suited to the 6809E because of its external clock requirements and its fixed memory access cycle time.

When implementing an interleaved memory access design, several points of concern need to be addressed. One major point is device access time. As an example, a typical non interleaved system operating at a clock speed of two megahertz will require memory and I/O controls with access times of 350ns or less. If we were to interleave this system, our access time would decrease to 210ns or less. This access time is not a problem for today's ram devices but it does present a problem for some eproms and most I/O controls. It is possible to find eproms with access times below 210ns but the standard two megahertz I/O controls for the MC6800 family have a minimum access time of 220ns which would be too slow for this design. To get around this problem, most interleaved systems will only allow interleaved access to ram or will restrict its clock speed to somewhere under two megahertz.

To avoid the access time problem associated with interleaved systems, the GT only allows interleaved access to its ram memory. To further simplify its design, the clock speed of the GT has been established at 1.5 megahertz. This allows the use of standard two megahertz I/O controls, a 250ns eprom and a 150ns ram.

The ram memory addresses of the GT are interleaved between the 68B09E processor and the 6545A video controller chip. The multiplexing of the addresses are handled by four 74LS157 multiplexor chips. The systems E clock signal is used to select the source for the multiplexing circuits. During the time when the video controller has access to ram, the processors data bus is isolated from the rams data output lines. This is accomplished by the 74LS245 bidirectional buffer chip. Once the ram has been addressed by the video controller, a shift register is loaded from the rams data output after the rams access time has been met. This data is then shifted out of the 74LS165 shift register at the rate of 12 megahertz, each data bit representing one pixel. During this access period it is necessary that the rams R/W signal remain in the read state condition.

In order to create a video image, in addition to pixel information we must provide synchronization signals for the video display device. These synchronization signals both horizontal and vertical are generated by the 6545A video controller chip. The 6545A also generates cursor and video enable signals which are all latched into a 74LS75 at the same time that the shift register is being loaded with ram data. The latched data and the shift registers pixel information is then combined to create a composite video signal that is suitable for our display device.

It is imperative that all of the before mentioned steps occur at precisely the same point in time. A time variation of only 20ns will be noticeable on the display screen. Because of this critical timing it is necessary to compensate for differences in propagation delays of the different chips being used. This is accomplished by using multiple buffer gates in series to delay certain critical signals such as the latch enable signal. The appearance of these buffers may appear redundant in the schematic but they are necessary for proper operation of the GT. It is also advisable to use only the "LS" versions of the 7400 series integrated circuits. Substituting non "LS" devices with different propagation delays will probably affect the quality of the display image.

The clock generation and timing signals required by the GT originate from a 12 megahertz crystal oscillator. I chose to use the oscillator as opposed to a discrete crystal in order to reduce the component count of the GT. The frequency of 12 megahertz is arrived at by the following

explanation. A standard video monitor has a horizontal scan time of 63.5us. Approximately 80% of this time can be used to display video information. The remaining 20% of this time will be used for horizontal sync and blanking. The amount of time allowed for display data is therefore 50.8us.

Since each display character frame is eight pixels wide and we wish to display 80 characters per line, we are therefore allowed a time of 79ns per pixel. This gives us a clock frequency of 12.6 megahertz. Since a 12.6 mhz oscillator would be hard to find, I chose the closest standard value of 12 mhz. This means that we will be displaying data for 84% of each horizontal scan time as opposed to 80%. This increase in display time may require a slight adjustment of the monitors horizontal width control in order to display all 80 characters.

The oscillators output is fed directly into the shift register and is used to clock the data bits out to the display. This clock is also fed into a 74LS193 counter who's outputs are decoded by a 74LS138 into eight individual state signals. Four of these state signals are used to set and reset two flip-flops which generate the systems E and Q clock signals. Another flip-flop is used to generate a ram enable signal which in effect disables the ram for a short period of time to allow for multiplexor address stabilization during the select transition time.

The state three signal is used to load the shift register and to latch the control signals coming out of the video controller chip. Notice that the clock signal driving the counter is delayed by two inverters in relation to the shift registers clock and also the state three signal which latches the video control signals is delayed in relation to the shift registers load signal. These delays are needed to compensate for differences in propagation delays which would affect the quality of the GT's display.

Control of the ram data buffer and the ram R/W signal is derived from the inverted system E clock signal, the processors address 15 line and the processors R/W signal. Decoding of the RAM, PIA, ACIA and VID enable signals are handled by a second 74LS138 decoder chip who's outputs are only enabled during the high portion of the systems E clock signal. Notice that the lower four outputs of the decoder are not used. These four outputs correspond to the memory locations occupied by the ram and are not needed since the ram buffer is already being controlled by the processors A15 line and the systems E clock signal.

The ACIA control used in the GT is a 6551A. This controller has an on board baud rate generator which can be controlled by software. This allows for changing the terminals baud rate without having to physically alter the circuit. The MAX232 driver chip is a fairly new device on the market. With the addition of four capacitors, the MAX232 can generate the positive and negative RS232 signal levels from a single five volt supply. This greatly reduces the total number of components required for RS232 compatibility.

The PIA control serves several functions in the GT. The A side data port of the PIA is used to read the configuration switch settings. The CA2 signal is used as an output which triggers a one shot which sounds the piezoelectric beeper. The B side of the PIA is used to access and control the keyboard.

The keyboard used with the GT must be an IBM XT compatible keyboard. This type of keyboard transmits its data in serial fashion. Each scan code received from the keyboard is eight bits in length and is preceded by one start bit. The start bit and data bits are shifted out of the keyboard in synchronization with a keyboard supplied clock signal. The data bits are valid during the fall time of each clock cycle. The 74LS164 shift register, flip-flop and inverter make up a nine bit shift register. When the last data bit is shifted into the 74LS164, the start bit is shifted into the flip-flop. This resets the flip-flops Q not output which is connected to the PIA's CB1 input. This transition of the CB1 input causes the PIA to generate a processor interrupt request. When the processor services the interrupt it first reads the B side data port of the PIA which will contain the keyboard scan code received by the shift register. The processor then toggles the CB2 output line low and then high which resets the shift register and flip-flop in anticipation of the next scan code to be received. It is important that the processor clears the shift register and flip-flop before the next scan code begins to shift in. If not, the GT would loose synchronization with the keyboard and only garbage would be received. Because the 68B09E can service interrupts much faster than the keyboard can generate them, this is not a real problem.

Well, that pretty much covers the theory behind the GT's hardware design. It is not necessary to understand all of the details involved with the GT's design in order to build a working terminal. I just thought that the explanation would be of interest to those who may want to modify or expand upon the basic design of the GT.

Construction

A picture of the GT's circuit board layout is included with this article. The picture is not to scale, however the grid markings on the board are one tenth of an inch on center. The construction technique I recommend is wire wrapping. For those of you not familiar with this type of construction, I will try to give you a few quick pointers.

First of all, the board must be cut to size before any components can be mounted. After cutting the board to size you should then drill the eight mounting holes and the one additional hole needed to attach the regulator and heat sink to the board. Notice the three holes across the center of the board are drilled between the existing prepunched grid holes.

Once drilled, I would suggest laying the board on a one inch thick piece of styrofoam insulation so that the discrete component mounting pins can be easily inserted. I recommend using Vector T-44 type pins. These pins need to be inserted wherever discrete components are to be mounted, such as by-pass capacitors, resistors, diodes etc. Once the pins are inserted you may then solder in all of the discrete components.

The next step after removing the board from the styrofoam base is to mount the cable connectors and the beeper. I prefer to use quick setting epoxy for this. Next insert all of the IC sockets but do not glue or fasten them to the board. The wire wrapped on the underside of the sockets will hold them in place.

Before you start to wire wrap, label the components and IC sockets on the underside of the board. Remember that pin one on a IC will vary depending on how you view the chip (top or bottom). Now you can start wrapping. I prefer to use pre-cut, pre-striped wire but thats entirely up to you and your budget. Each time you wrap a wire, mark the wire on the schematic with a yellow magic marker. This will insure that you do not miss any connections. Because the GT is fairly small and has few integrated circuits, I decided not to run solid Vcc and ground busses. You should wrap the Vcc and ground wires first before any other signals and you may want to double up on the ground grid to insure noise free operation.

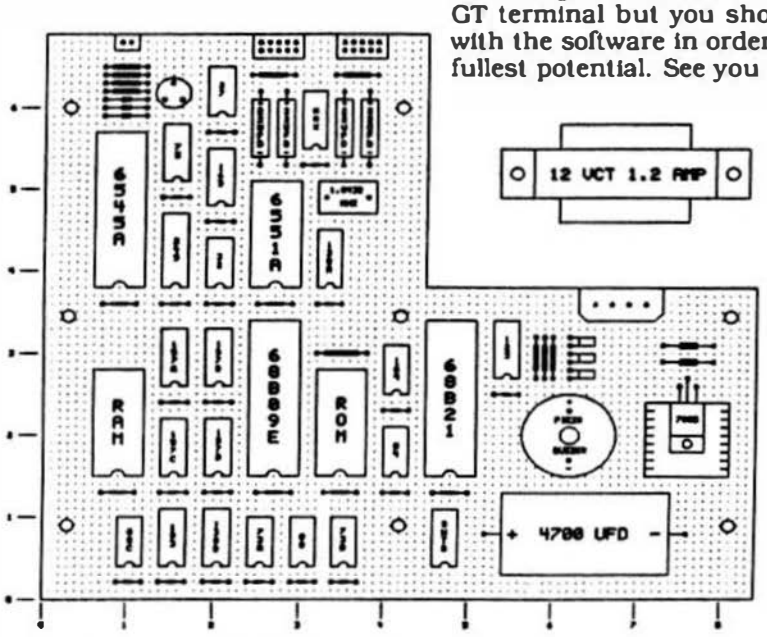
After the wire wrapping is complete you will need to mount the board in the case. You will have to use one half inch spacers to provide clearance for the wire wrap pins. I wont go into detail as to how or where you should mount the power transformer, plugs, switch, fuse holder, etc. Keep in mind when you are wiring for 120 volts ac to be careful and provide proper clearance where needed.

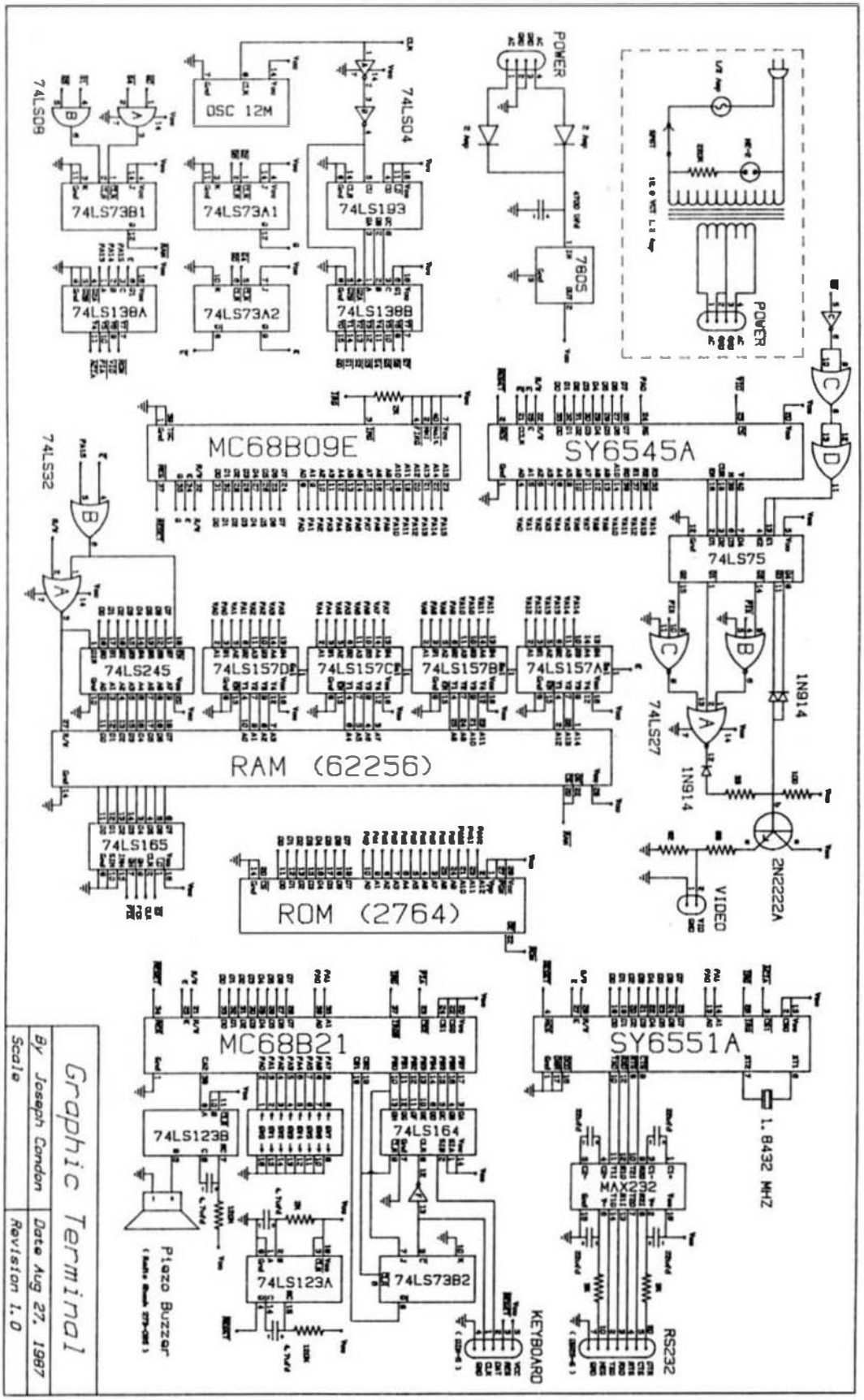
The last issue I want to discuss is parts and where to get them. All the components for the GT can be purchased from Jameco Electronics except for a few odds and ends like the piezoelectric buzzer and power transformer which you can pick up at your local Radio Shack store. All of the parts used in the GT are standard off the shelf components and can be purchased just about anywhere. You may want to shop around for the best prices but I think you will find Jameco Electronics hard to beat for this project.

As for the keyboard, I use a JE1015 from Jameco and it must be used in the XT mode of operation. I have not tried any other keyboards on the GT but if it is XT compatible, it should work. The monitor I use is a NEC JB-1201MA character display. If you shop around, you should be able to pick one up for about $80.00. Just about any standard composite video monitor should work with the GT.

Although any XT compatible keyboard and any composite video monitor should operate correctly with the GT, I cannot guarantee that they will. All I know for a fact is that the JE1015 keyboard and the JB-1201MA monitor work properly with the GT.

Well, that about does it for the hardware portion of the GT. Next month we can all change hats and become programmers. As I mentioned at the beginning of this article, you don't have to be a programmer in order to build a functioning GT terminal but you should at least be familiar with the software in order to exploit the GT to it's fullest potential. See you next month.




Graphic Terminal
By Joseph Condon
Date Aug 27, 1987
Scale
Revision 1.0
OSC 12M
74LS04
74LS09
74LS73B1
74LS73A1
74LS193
74LS138A
74LS73A2
74LS138B
POWER
7805
MC68B09E
SY6545A
74LS32
74LS75
74LS245
74LS157D
74LS157C
74LS157B
74LS157A
74LS27
1N914
RAM (62256)
74LS165
ROM (2764)
VIDEO
2N2222A
MC68B21
SY6551A
74LS123B
74LS164
1.8432 MHZ
MAX232
Piezo Buzzer
74LS123A
74LS73B2
KEYBOARD
RS232

Bit-Bucket

By: All of us

"Contribute Nothing · Expect Nothing", DMW '86

Continued from Sept. 87

A PL/9 Interface for ISAM

by: Martin C Gregorie
10 Sadlers Mead
Harlow, Essex U.K.

```
/* sets up an icb parameter ptr/lth pair */

procedure isamparam(integer .icb,n,.ptr,lth):
   integer i;
   i=n*2;
   icb(i)=.ptr;
   icb(i+1)=lth;
endproc;

/* calculate the length of a string */

procedure isam_lth(byte .str):
   integer n;
   n=0;
   while str(n) n=n+1;
endproc n;

/* copy bytes converting to upper case (file spec) */

procedure isamcopy(byte .s,.d: integer length):
   byte    c;
   integer i;
   i=0;
   while i<length
      begin
         c=s(i);
         if c>='a .and c<='z then c=c-$20;
         d(i)=c;
         i=i+1;
      end;
   d(i)=0;
endproc .d;

/* deduce an ISAM command from the .cmd parameter */

procedure isampos(byte .cmd):
   integer r;
   if !.cmd<.first then      /* unsigned ! */
      r=.istrt
   else
      r=!.ispos+.cmd-.first;
endproc integer r;

/* clear the icb */

procedure isamclear(integer .icb):
   integer i;
   i=0;
   repeat
      icb(i)=0;
      i=i+1;
   until i>24;
endproc;

/* common setup code for ready and format */

procedure isamsetup(integer .icb, id: byte .action, .name:
                    integer rlen,klen);
   isamclear(.icb);
   icb(13)=id;
   isamparam(.icb,0,.icb(13),2);
   isamparam(.icb,1,.action,2);
   isamparam(.icb,2,isamcopy(.name,.icb(14),isam_lth(.name)),
             isam_lth(.name));
   icb(24)=rlen;
   isamparam(.icb,3,.icb(24),2);
   icb(22)=klen;
endproc;

/* compare a key with a record */

procedure match(byte .key,.rec: integer lth):
   integer i,r;
   i=0;
   r=0;
   while i<lth .and key(i) .and rec(i)
      begin
         if key(i)<>rec(i) then r=-100; /* key mismatch */
         i=i+1;
      end;
endproc r;

/*
   User interface procedures
   -------------------------
*/

procedure dbstatus(integer .icb);
endproc integer icb(12);

procedure ready(integer .icb, id: byte .name; integer rlen,klen);
   isamsetup(.icb,id,.isrdy,.name,rlen,klen);
   isam(.icb);
endproc dbstatus(.icb);

procedure finish(integer .icb);
   isamparam(.icb,1,.isfin,2);
   isam(.icb);
endproc dbstatus(.icb);

procedure abort(integer .icb);
   isamparam(.icb,1,.isabt,2);
   isam(.icb);
endproc dbstatus(.icb);

procedure position(integer .icb: byte .cmd);
   integer x;
   x=isampos(.cmd);
   isamparam(.icb,1,x,2);
   if x=.istrt then
      isamparam(.icb,5,.cmd,isam_lth(.cmd));
   if .cmd=.first .or .cmd=.last .or x=.istrt then
      isam(.icb)
   else
      icb(12)=201;          /* reply 201=wrong verb */
endproc dbstatus(.icb);

procedure obtain(integer .icb: byte .cmd,.rec):
   integer ptr;
   icb(12)=0;
   ptr=isampos(.cmd);
   if .cmd=.first .or .cmd=.last .or ptr=.istrt then
      position(.icb,.cmd);
   if dbstatus(.icb)=0 then
      begin
         if .cmd=.last then
            ptr=isampos(.previous);
         if .cmd=.first .or ptr=.istrt then
            ptr=isampos(.next);
         isamparam(.icb,1,ptr,2);
         isamparam(.icb,5,.rec,icb(24));
         isam(.icb);
      end;
   if !.cmd<.first then
      if dbstatus(.icb)=0 then
         icb(12)=match(.cmd,.rec,icb(22)); /* the right record? */
endproc dbstatus(.icb);

procedure store(integer .icb: byte .rec);
   isamparam(.icb,1,.isadd,2);
   isamparam(.icb,5,.rec,icb(24));
   isam(.icb);
endproc dbstatus(.icb);

procedure erase(integer .icb);
   isamparam(.icb,1,.isdel,2);
   isam(.icb);
endproc dbstatus(.icb);

procedure modify(integer .icb: byte .rec);
   erase(.icb);
   if dbstatus(.icb)=0 then
      store(.icb,.rec);
endproc dbstatus(.icb);

procedure reorganise(integer .icb);
   isamparam(.icb,1,.isreo,2);
   isam(.icb);
endproc dbstatus(.icb);

procedure format(integer .icb, id: byte .name;
                 integer rcnt,rlen,klen);
   isamsetup(.icb,id,.ircrt,.name,rlen,klen);
   icb(23)=rcnt;
   isamparam(.icb,4,.icb(23),2);
   isam(.icb);
endproc dbstatus(.icb);
```

EOF

Dear Mr. Williams,

Last month I was pleased to discover the existence of your journal. Like most of your readers, I too am a die-hard. Back when IBM stormed the country, I was angered that Big Blue chose Intel processors for their line of personal computers. Everyone who reads your journal knows that Motorola makes a superior microprocessor, but you know the market. It is like a swarm of lemmings that all follow IBM like it is the only computer company. Don't get me wrong. Good things came form the PC/XT and PC/AT. Five years ago I never would have dreamed that I could buy a Mbyte hard disk for less than $300.00. They are pretty good machines. Everyone knows that it could have been a much better machine if it had started out with an MC68000 engine under its hood.

But even though I am still mad at IBM to this day, I find relief in discovering your journal. Your subscribers are a rare group of people just like me. It is lonely out here. Most of the people I know who are interested in computers are either Display-Write Grunts or Lotus Zombies. That is little better than being completely computer illiterate. The days of the computer hobbyist seem to be dwindling (even though "we" built the industry). But I know there is hope. I must admit, I own an IBM clone. But I also have an MC68XXX software development package form SDS Inc. which I have big plans for. I am hoping that a few of us die-hards can get our heads together and make a big splash in the micro-computer industry.

Just a little biographical information. I have been a computer hobbyist ever since I dropped out of graduate school in wildlife biology in 1979 (and abandoned my Master's Thesis on Lizard Ecology and Behavior!). After doing laboratory analyst work and being unemployed for several years I am now a computer systems analyst in the chemical manufacturing industry. This September will mark my third year of computer work for my company. I now have a hand-full of systems running out in the plant. I have many very promising ideas and some of those will be filed with the U.S Patent Office. My experience is primarily with software but I am very interested in hardware too. I would like to use your journal to reach others like myself who are either amateurs or professionals with big commercial goals. If any of your readers are interested in developing and marketing some ideas (using Motorola parts of course) have them write to me at:

Advanced Process Control
Rohm & Haas Texas, Inc.
Deer Park, Texas 77536

Enclosed is a check for a 3 year subscription to 68 MICRO JOURNAL. Thank you.

Sincerely,

Paul K. Mckneely

GESPAC, INC

50 West Hoover Ave.
Mesa, Arizona 85202
Tel. (602) 962-5559
Fax. (602) 962-5750

Reader Contact: Mark Stephens
Editorial Contact: Cosma Pabouctsidis

WESCON BOOTH: 5607

NEW FAMILY OF INDUSTRIAL MICROCOMPUTERS PERFORMS IN HARSH ENVIRONMENT.

MESA, AZ, September 25, 1987--GESPAC announces its entry in the systems business with the introduction of a new family of versatile microcomputer systems aimed at the industrial marketplace. The GESCOMP is a powerful real-time, multi-tasking microcomputer system in a rugged and compact single height Eurocard package, that allows the user to monitor, test and automate processes in a variety of harsh industrial and military environments.

The GESCOMP system is intended to be used as a process or cell controller in factory automation applications. The GESCOMP is ideal for use as imbedded computers in intelligent machine tools and specialized instrumentations. The GESCOMP systems can also be used as total hardware and software development workstations.

The GESCOMP offers the highest level of processing power for systems of its class and size. At the low end of GESPAC's offering is the GESCOMP 8300-M which uses an 8 MHz 16-bit 68000 microprocessor and comes equipped with two 1 Megabyte 3.5" floppy disk drives and 512 Kilobytes of RAM. The most powerful member of the GESCOMP family is the GESCOMP 8340-P/HT which features a very high performance 32-bit 68020 microprocessor running at 16.7 MHz with a 68881 arithmetic co-processor. 2.5 Megabytes of RAM, 1 Megabyte of 3.5" floppy disk storage and 40 Megabytes of hard disk storage. Several variations of these systems are offered by GESPAC depending on the amount of processing power needed and the required disk/RAM storage capacity. The open G-64 bus architecture of the systems allows memory expansions up to 32 Megabytes.

The GESCOMP system is supported by the OS-9 disk operating system which is well accepted and familiar to the automation industry. OS-9 features an advanced "UNIX-Like" structure and form. A special library allows OS-9 to run programs written for UNIX in C. Unlike UNIX, however, OS-9 features a lean and fast modular structure that can easily be put into ROM for diskless systems. The real-time, multi-tasking nature of OS-9 permits the user to divide his applications into several concurrent tasks, while allowing real-time response to outside events. The modular, multi-user architecture of OS-9 permits the addition of a new user to the system in no time. Depending on the load of work, the GESCOMP system will support up to 16 users on line simultaneously.

The GESCOMP is delivered complete with the disk operating system, a screen editor, macro assembler, symbolic debugger, linker, and a C compiler. Other high level languages interpreters or compilers are optionally available for Fortran, Forth, Basic and Pascal.

A key feature of the GESCOMP system is its open G-64 bus architecture which allows a degree of customization which is unsurpassed in the industry. This customization is essential in order to meet the specific requirements of most industrial applications. For instance, a GESCOMP may be configured as a data acquisition system by adding analog I/O modules and additional nonvolatile storage memory. Or, by adding motor controller cards and industrial I/O interfaces, the GESCOMP can be the main controlling unit of a milling machine.

A GESCOMP system typically provides 8 unused slots, each of which can accept any one of the 150 G-64 board level products in GESPAC's catalog. The GESCOMP will also accept G-64 bus board level products from any of the more than 30 independent G-64 bus vendors. The G-64 bus is a second generation microcomputer architecture aimed at midrange industrial applications.

GESCOMP systems are designed to communicate with the operator with a standard CRT terminal. It is also possible to connect graphics cards into the system to display high resolution pictorial information of 640 by 480 pixels to 1024 by 1024 pixels in 256 colors out of a palette of 262,144. The user can easily install one or more of these graphics controllers into the same backplane to simultaneously control multiple displays.

Another unique feature of the GESCOMP systems is their networking capability. Using GESNET, a proprietary network protocol from GESPAC, up to 50 GESCOMP systems can talk to one another over 3,000 feet of coaxial cable. The Network operates at 800 kilobits per second and achieves very high throughput thanks to its superior CSMA/CA (Collision Sense Multiple Access / Collision Avoidance) arbitration and its Direct Memory Access operation. GESNET's cost per node is a fraction of the typical cost of more publicized networks such as Ethernet or MAP.

A version of the GESNET controller board is to be released by GESPAC for the VME bus. This connection will allow GESCOMP systems to perform as a front end processor with VME based supervisors in a hierarchical, distributed processing architecture. The network is totally integrated into the disk operating system, thus allowing transparent access to all resources. For instance, all graphic screens, disk storage and communications ports are accessible from any processor in the network.

For use in the most severe environments, where mechanical disk drives are not allowed, the GESCOMP can run without a disk. In this mode of operation, the systems can boot from the operating systems and application programs located in EPROM. The GESCOMP is also expandable to use a bubble memory cartridge system for use in these environments. Using GESNET, a disk based GESCOMP located in a clean environment can serve as file server for several ROM based GESCOMP systems on the harsh factory floor.

The GESCOMP system uses a modular architecture with all vital functions readily accessible behind the front panel for easy service and expansion. The system is based on the 100 by 160 millimeter (4" x 6.25") G-64 bus Eurocard and features a rugged DIN 41612 pin-in-socket backplane architecture. The small form factor of the cards and the superior DIN connector, make the GESCOMP particularly resistant to shock, vibration, and corrosion due to airborne contaminates. GESCOMP is packaged in a table enclosure for development or laboratory environment or in a 19" rack for mounting in a NEMA enclosure or directly into the application. Each GESCOMP system includes a 200 Watt power supply.

Other versions of the GESCOMP systems, using the 80286 microprocessor and the MS-DOS operating systems, are also available from GESPAC.

GESCOMP systems are available now, prices start at $3995 for single quantity orders of the basic system configuration. OEM discounts are available for large quantity orders.

• • •

Classifieds — As Sunmitted - No Guarantees

AT&T 7300 UNIX PC, UNIX V OS, 1MB Memory, 20 MB Hard Disk, 5" Drive, Internal Modem, Mouse. Best Offer Gets It.

S+ System with Cabinet, 20 Meg Hard Disk & 8" Disk Drive with DMAF3 Controller Board. 1-X12 Terminal $4800.

DAISY WHEEL PRINTERS
Qume Sprint 9 - $900
Qume Sprint 5 - $800.

HARD DISK 10 Megabyte Drive - Seagate Model #412 $275.

3 - Dual 8" drive enclosure with power supply . New in box . $125 each.

5 - Siemens 8" Disk Drive , $100 each.

Tano Outpost II, 56K, 2 5" DSDD Drives, FLEX, MUMPS, $495.

TELETYPE Model 43 PRINTER - with serial (RS232) interface and full ASCII keyboard . $250 ready to run.

SWTPC S/09 with Motorola 128K RAM, 1-MPS2, 1-Paralell Port, MP-09CPU Card - $900 complete.

Tom (615) 842-4600 M-F 9AM to 5PM EST

•••

I Want To Buy
GIMIX #68 DMA FDC. Price and details to
Roger Steedman
RMB 9010 OMEO Hwy
Sarsfield 3883, Australia

•••

PT68K-1A (SameSBC as Mustang-08/A). 12 MHz 68008,768 K RAM. 2-80 Track, 25M Hard Disk. $1500
Martin Bose (415) 351-7297

•••

SWPTC 6809 with DMAF2 Subsystem, two Serial Ports, Televideo 950, NEC 8023A Printer, Milgo 300/1200 modem. All fot $900.
Al (214) 262-1286 - 7pm to 10pm CST

THE 6800-6809 BOOKS

..HEAR YE.....HEAR

OS-9™ User Notes

By: Peter Dibble

The publishers of 68' Micro Journal are proud to make available the publication of Peter Dibbles **OS9 USER NOTES**

Information for the BEGINNER to the PRO, Regular or CoCo OS9

Using OS9
HELP, HINTS, PROBLEMS, REVIEWS, SUGGESTIONS, COMPLAINTS, OS9 STANDARDS, Generating a New Bootstrap, Building a new System Disk, OS9 Users Group, etc.

Program interfacing to OS9
DEVICE DESCRIPTORS, DIRECTORIES, "FORKS", PROTECTION, "SUSPEND STATE", "PIPES", "INPUT/OUTPUT SYSTEM", etc.

Programming Languages
Assembly Language Programs and Interfacing: Basic09, C, Pascal, and Cobol reviews, programs, and uses; etc.

Disks Include
No typing all the Source Listings in. Source Code and, where applicable, assembled or compiled **Operating Programs.** The Source and the Discussions in the Columns can be used "as is", or as a "Starting Point" for developing your OWN more powerful Programs. Programs sometimes use multiple Languages such as a short Assembly Language Routine for reading a Directory, which is then "piped" to a Basic09 Routine for output formatting, etc.

BOOK $9.95

Typeset -- w/ Source Listings
(3-Hole Punched; 8 x 11)
Deluxe Binder - - - - - - - - - - $5.50

All Source Listings on Disk

1-8" SS, SD Disk - - - - $14.95
2-5" SS, DD Disks - - - $24.95

FLEX™ USER NOTES

By: Ronald Anderson

The publishers of 68 MICRO JOURNAL are proud to make available the publication of Ron Anderson's **FLEX USER NOTES**, in book form. This popular monthly column has been a regular feature in 68' MICRO JOURNAL SINCE 1979. It has earned the respect of thousands of 68 MICRO JOURNAL readers over the years. In fact, Ron's column has been described as the 'Bible' for 68XX users, by some of the world's leading microprocesaor professionals. The most needed and popular 68XX book available. Over the years Ron's column has been one of the most popular in 68 MICRO JOURNAL. And of course 68 MICRO JOURNAL is the most popular 68XX magazine published.

Listed below are a few of the **TEXT** files included in the book and on diskette.

All TEXT files in the book are on the disks.

LOGO.C1	File load program to offset memory — ASM PIC
MEMOVE.C1	Memory move program — ASM PIC
DUMP.C1	Printer dump program — uses LOGO — ASM PIC
SUBTEST.C1	Simulation of 6800 code to 6809, show differences — ASM
TERMEM.C2	Modem input to disk (or other port input to disk) — ASM
M.C2	Output a file to modem (or another port) — ASM
PRINT.C3	Parallel (enhanced) printer driver — ASM
MODEM.C2	TTL output to CRT and modem (or other port) — ASM
SCIPKG.C1	Scientific math routines — PASCAL
U.C4	Mini-monitor, disk resident, many useful functions — ASM
PRINT.C4	Parallel printer driver, without PFLAG — ASM
SET.C5	Set printer modes — ASM
SETBAS1.C5	Set printer modes — A-BASIC

NOTE: .C1,.C2, etc.=Chapter 1, Chapter 2, etc.

**Over 30 TEXT files included is ASM (assembler)-PASCAL-PIC (position independent code) TSC BASIC-C, etc.

Book only: **$7.95** + $2.50 S/H

With disk: 5" **$20.90** + $2.50 S/H

With disk: 8" **$22.90** + $2.50 S/H

Shipping & Handling $3.50 per Book, $2.50 per Disk set
Foreign Orders Add $4.50 Surface Mail
or $7.00 Air Mail
If paying by check - Please allow 4-6 weeks delivery
*** All Currency in U.S. Dollars**

Continually Updated In 68 Micro Journal Monthly

Computer Publishing Inc.
5900 Cassandra Smith Rd.
Hixson, TN 37343

(615) 842-4601
Telex 5106006630

"FLEX is a trademark of Technical Systems Consultants
"OS9 is a trademark of Microware and Motorola
"68' Micro Journal is a trademark of Computer Publishing Inc.

THE GMX 020BUG DEBUGGER/DIAGNOSTIC PACKAGE

This extensive firmware package provides a broad range of program development tools and a complete suite of diagnostic programs for exercising GMX Micro-20 hardware.

The debugger includes commands for displaying and modifying registers and memory. If the optional 68881 Floating-Point Coprocessor is installed, its registers are also accessible. Memory can be displayed in hexadecimal and ASCII format, as floating-point values (single, double, extended or packed format), or as disassembled instructions (including FPC instructions). Memory modify can be done with hexadecimal values, with ASCII strings, with floating-point values, or with a one-line assembler which supports the full 68020 instruction set (although not the FPC instructions). Block move, fill, and search are also available.

Several different modes for tracing or executing user programs are provided, along with a powerful breakpoint facility. Programs and data may be downloaded from a host system or uploaded back to the host, and the GMX Micro-20 console may be used as a host system terminal. A serial printer may be hooked up, and used to make hardcopy listings of debugger sessions as desired.

The diagnostic firmware includes 90 test commands and 16 utilities. Complete test suites are provided for each functional block of the GMX Micro-20's hardware, including, for example, 9 different tests for memory, 9 tests for serial I/O ports, 2 tests for the 68881 FPC, and 9 tests for the optional memory management unit. For the peripheral control interfaces (floppy disk, SASI/SCSI hard disk or tape), test commands support a broad range of peripheral operations (read, write, format, etc.) so that the user may test both the interface and an attached device. Tests are provided for add-on I/O boards, including the ARCnet interface, I/O Channel interface, and parallel and serial expansion boards.

The utility commands allow the user to execute groups of test commands conveniently, repeat commands or command groups, enable or disable detailed fault reporting, count detected errors, or execute all the non-peripheral tests as a group. A switch option allows this last function to be invoked automatically at power-on or reset. Other utilities allow the user to check the state of the various jumpers and switches on the GMX Micro-20 directly.

In addition to the Diagnostic command package, 020Bug contains a confidence test which is always run after power-on or reset. This test does a quick checkout of the processor and the basic system elements that are needed for 020Bug operation. If any defect is found, an error code is signalled by on-and-off blinks of an LED.

DEBUGGING COMMANDS

MD — Memory display
MM — Memory modify
MS — Memory set
BF — Block fill
BM — Block move
BS — Block search
RD — Register display
RM — Register modify
OF — Offset registers
BR — Breakpoint set
NOBR — Breakpoint delete
G — Go to target code
GD — Go, delete breakpoints
GN — Go, stop after 1 instruction
GT — Go, set temp breakpoint
T — Trace
TC — Trace on change of flow
TT — Trace to temp breakpoint
LO — Download
OU — Upload
VE — Verify download
TM — Terminal mode
PA — Printer attach
NOPA — Remove printer
PF — Port format
TD — Time display
TS — Time set
SD — Switch directory
RS — Restart system
OS — Boot operating system

UTILITY COMMANDS

NV — Non-verbose mode
SE — Stop on error mode
LE — Loop on error mode
LC — Loop continual mode
ST — Selftest mode
STL — Selftest with LED mode
DE — Display errors
ZE — Zero errors
DP — Display pass count
ZP — Zero pass count
RL — Read loop
WL — Write loop
DJ — Display baud rate jumper settings
DS — Display switch
MJ — Display MMU board jumper settings
SX — Scan I/O expansion space

TEST COMMANDS

AN — ARCnet interface tests
- A — Wakeup test
- B — DIP Switch test
- C — Interrupts test
- D — Buffer test

CA20 - On chip cache tests
- A — Basic caching
- B — Unlike function codes
- C — Disable
- D — Clear

FD — Floppy disk tests
- A — Set parameters
- B — Drive select toggle
- C — Side select toggle
- D — Restore
- E — Seek
- F — Format track
- G — Read
- H — Write
- I — Copy buffer
- J — Compare buffer
- K — Fill buffer

IC — I/O Channel tests
- A — Print test pattern
- B — Bit rotate

MH — Miscellaneous hardware tests
- A — 68881 FPC instructions
- B — 68881 FPC control functions
- C — Tick generator
- D — Interrupt sources

MT — Memory tests
- A — Set function code
- B — Set start address
- C — Set end address
- D — Random inversion test
- E — March address test
- F — Walk-a-bit test
- G — Refresh test
- H — Random byte test
- I — Program test
- J — TAS test
- K — Test 0000-1FFF
- L — Partial longword writes test

MU — Memory Management tests
- A — Map RAM data test
- B — Map RAM address test
- C — Map RAM partial write test
- D — Map RAM random data test
- E — Accessed bit reset test
- F — Address mapping test
- G — Accessed/Dirty bits test
- H — Valid/Write Enable test
- I — Task size test

PP — Parallel port tests
- A — Print test pattern
- B — Continual test bit pattern
- C — Test bit pattern for 10 sec

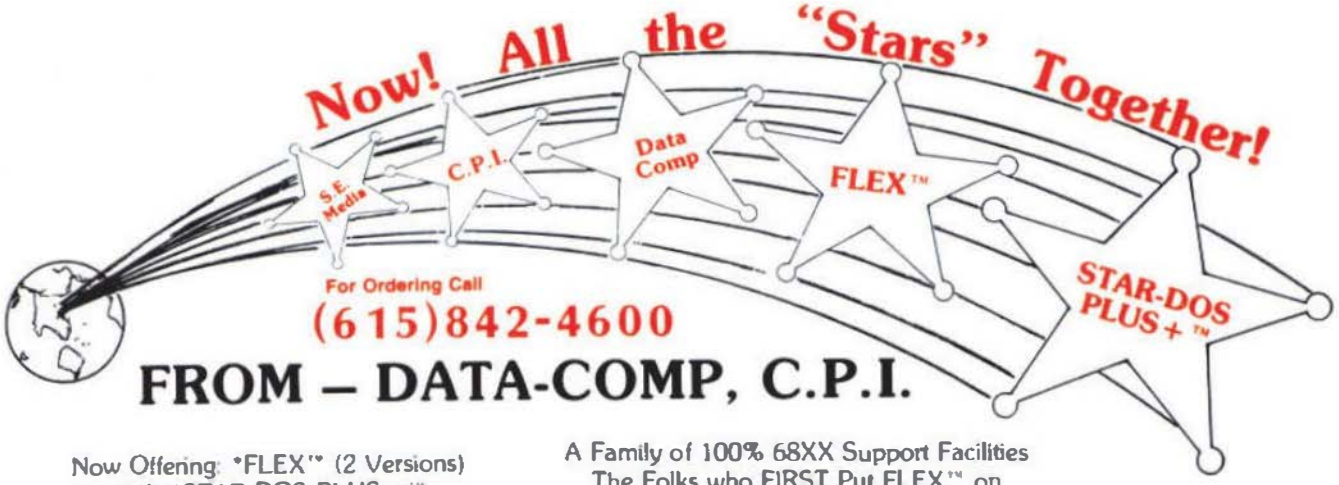
PX — Parallel I/O expansion board tests
- A — Data, handshake, and IRD test
- B — P4 connector test
- C — Data and handshake toggle

SA — SASI/SCSI port with SASI device
- A — Select drive
- B — Scan data lines
- C — Restore
- D — Seek
- E — Read
- F — Write
- G — Compare buffers
- H — Fill write buffer
- I — Test interrupt
- J — Park head
- K — Format

SC — SASI/SCSI port with SCSI device
- A — Select drive
- B — Scan data lines
- C — Restore
- D — Seek
- E — Read
- F — Write
- G — Compare buffers
- H — Fill write buffer
- I — Test interrupt
- J — Stop drive
- K — Format

SI — Serial I/O tests
- A — Select DUARTs
- B — Internal loopback
- C — External loopback
- D — Baud rates
- E — Parity modes
- F — Character lengths
- G — Handshake lines
- I — BREAK detect
- J — Interrupt output
- K — Continual handshake toggle

TA — Tape drive tests
- A — Rewind
- B — Read
- C — Write
- E — Compare buffers
- F — Fill write buffer
- G — Erase



FROM – DATA-COMP, C.P.I.

Now Offering: *FLEX™ (2 Versions)
AND *STAR-DOS PLUS+™

A Family of 100% 68XX Support Facilities
The Folks who FIRST Put FLEX™ on
The CoCo

FLEX-CoCo Sr.
with TSC Editor
TSC Assembler
Complete with Manuals
Reg. $250.00 **Only $79.95**

STAR-DOS PLUS+
- Functions Same as FLEX
- Reads - writes FLEX Disks
- Run FLEX Programs
- Just type: Run "STAR-DOS"
- Over 300 utilities & programs to choose from.

$34.95

FLEX-CoCo Jr.
without TSC Editor & Assembler
$49.95

PLUS

ALL VERSIONS OF FLEX & STAR-DOS• INCLUDE

+ Read-Write-Dir RS Disk
+ Run RS Basic from Both
+ More Free Utilities
+ External Terminal Program
+ Test Disk Program
+ Disk Examine & Repair Program
+ Memory Examine Program
+ Many Many More!!!

TSC Editor
Reg $50.00
NOW $35.00

TSC Assembler
Reg $50.00
NOW $35.00

CoCo Disk Drive Systems

2 THINLINE DOUBLE SIDED DOUBLE DENSITY DISK DRIVES SYSTEM WITH POWER SUPPLY,CABINET,DISK DRIVE CABLE,J&M NEW DISK CONTROLLER JFD-CP WITH J-DOS,RS-DOS OPERATING SYSTEMS.	$469.95
* Specify What CONTROLLER You Want J&M, or RADIO SHACK	
THINLINE DOUBLE SIDED DOUBLE DENSITY 40 TRACKS	$129.95

Verbatim Diskettes

Single Sided Double Density	$ 24.00
Double Sided Double Density	$ 24.00

Controllers

J&M JFD-CP WITH J-DOS	$139.95
WITH J-DOS, RS-DOS	$159.95
RADIO SHACK 1.1	$134.95
RADIO SHACK Disk CONTROLLER 1.1	$134.95

Disk Drive Cables

Cable for One Drive	$ 19.95
Cable for Two Drives	$ 24.95

MISC

64K UPGRADE FOR C,D,E,P, AND COCO 11	$ 29.95
RADIO SHACK BASIC 1.2	$ 24.95
RADIO SHACK DISK BASIC 1.1	$ 24.95
DISK DRIVE CABINET FOR A SINGLE DRIVE	$ 49.95
DISK DRIVE CABINET FOR TWO THINLINE DRIVES	$ 69.95

PRINTERS

EPSON LX-80	$289.95
EPSON MX-70	$325.95
EPSON MX-100	$495.95

ACCESSORIES FOR EPSON

8148 2K SERIAL BOARD	$ 89.95
8149 32K EXPAND TO 128K	$169.95
EPSON MX-RX-80 RIBBONS	$ 7.95
EPSON LX-80 RIBBONS	$ 5.95
TRACTOR UNITS FOR LX-80	$ 39.95
CABLES & OTHER INTERFACES CALL FOR PRICING	

DATA-COMP
5900 Cassandra Smith Rd.
Hixson, TN 37343

MasterCard VISA

SHIPPING
USA ADD 2%
FOREIGN ADD 5%
MIN. $2.50

(615)842-4600
For Ordering
Telex 5106006630

An Ace of a System in Spades! The New

MUSTANG-08/A™

Now with 4 serial ports standard & speed increase to 12 Mhz CPU + on board battery backup and includes the PROFESSIONAL OS-9 package - including the $500.00 OS-9 C compiler! This offer won't last forever!

NOT 128K, *NOT* 512K
FULL 768K No Wait RAM

The MUSTANG-08™ system took every hand from all other 68008 systems we tested, running OS-9 68K!

The MUSTANG-08 includes OS9-68K™ and/or Peter Stark's SK*DOS™. SK*DOS is a single user, single tasking system that takes up where *FLEX™ left off. SK*DOS is actually a 68XXX FLEX type system *(Not a TSC product.)*

The OS-9 68K system is a full blown multi-user, multi-tasking 68XXX system. All the popular 68000 OS-9 software runs. It is a speed whiz on disk I/O. Fact is: the MUSTANG-08 is faster on disk access than some other 68XXX systems are on memory cache access. Now, that is fast! And that is just a small part of the story! See benchmarks.

System includes OS-9 68K or SK*DOS - Your Choice

Specifications:

CPU	MC68008	12 Mhz
RAM	768K No Wait States	256K Chips
PORTS	4 - RS232 2 - 8 bit Parallel	MC68681 DUART MC6821 PIA
CLOCK	MK48T02	Real Time Clock Bat. B/U
EPROM	16K, 32K or 64K	Selectable
FLOPPY	WD1772	5 1/4 Drives
HARD DISK	Interface Port	WD1002 Board

Now more serial ports - faster CPU Battery B/U - and $850.00 OS-9 Professional with C compiler included!

***$400.00** See Mustang-02 Ad - page 5 for trade-in details

Now even faster! with 12 Mhz CPU

C Compile times: OS-9 68K Hard Disk	
MUSTANG-08 8 Mhz CPU	0 min - 32 sec
Other popular 68008 system	1 min - 05 sec
MUSTANG-020	0 min - 21 sec

25 Megabyte Hard Disk System

$1,998.90

Complete with PROFESSIONAL OS-9 includes the $500.00 C compiler, PC style cabinet, heavy duty power supply, 5" DDDS 80 track floppy, 25 MegByte Hard Disk - Ready to Run

Unlike other 68008 systems there are several significant differences. The MUSTANG-08 is a full 12 Megahertz system. The RAM uses NO wait states, this means full bore MUSTANG type performance.

Also, allowing for addressable ROM/PROM the RAM is the maximum allowed for a 68008. The 68008 can only address a total of 1 Megabytes of RAM. The design allows all the RAM space (for all practical purposes) to be utilized. What is not available to the user is required and reserved for the system.

A RAM disk of 480K can be easily configured, leaving 288K free for program/system RAM space. The RAM DISK can be configured to any size your application requires (system must have 128K in addition to its other requirements). Leaving the remainder of the original 768K for program use. Sufficient source included (drivers, etc.)

FLEX is a trademark of TSC — MUSTANG-08 is a trademark of CPI

MUSTANG-08

LOOK

Mustang Hi-Speed Systems Only from Data-Comp Div. 68008-68030

				Seconds	32 bit Integer	Register Long
Other	68008	8 Mhz	OS-9	68K	18.0	9.0
MUSTANG-08	10 Mhz		OS-9	68K	9.8	6.3

```
Main()
{
                    /* int i; */
C Benchmark Loop    register long i;
                    for (i=0; i < 999999; ++i);
}
```

Data-Comp Division

A Decade of Quality Service™

Systems World-Wide

Computer Publishing, Inc. 5900 Cassandra Smith Road

Telephone 615 842-4601 - Telex 510 600-6630 Hixson, Tn 37343

* Those with SWTPC Hi-density FLEX 5" - Call for special info.